64 PLUS 4



& AMIGA

MAJ 1991 ISSN 0867-3918 INDEKS 377112 CENA 5000 zł

MIESIĘCZNIK UŻYTKOWNIKÓW KOMPUTERÓW COMMODORE



W NUMERZE:

- Własna dyskietka
- LEMINGS
- SCULPT 4D

W numerze :

W następnym numerze :

- **KICKSTART 2.0 - test**
- **Stacja 1541 i AMIGA**
- **JEWELS OF DARKNESS**
- **Cheats C-64**

OGŁOSZENIA

KOMPUTEROWA FIRMA USŁUGOWA "TREND" - COMMODORE AMIGA: oprogramowanie, literatura. Informacja: dyskietka lub koperta + znaczek. Kontakt: Rafał Wierzbicki, ul. Rogowska 86/10, 54-440 Wrocław.

C128, STACJA, MONITOR, MYSZ, AKCESORIA - SPRZEDAM - 3900000zł. Gorzów tel. 32-14-28.

C-64 Dużo programów taśma/dysk wymienię. Stanisław Pieniążek, ul. Konwaliowa 10/83, 81-651 Gdynia.

Wymienię oprogramowanie na C-64 (kaseta). Paweł Ciborowski, ul. Chopina 3/71, 18-403 Łomża.

C-64 klawiatura, obudowa, zasilacz nowe - sprzedam. D. Miszczak, ul. Kilińskiego 160/30, 90-322 Łódź.

Sprzedam dyski nagrane do AMIGI- 11tys/szt. Katowice tel. 531-792.

Programy C-64 (magnetofon) - duży wybór, krótkie terminy. Katalog - koperta, znaczek. Jacek Grzejszczak, 93-333 Łódź, ul. Ogniskowa 10/8.

Uwaga! MOJA AMIGA - książka autorska napisana prostym językiem. Informacje: M. Pampuch, Kraków, tel 0-12-471 761.

Poszukuję instrukcji i programów obsługujących MIDI (Amiga). Robert Dajczak ul. Sportowa 3/1, 66-100 Babimost.

GRY I PROGRAMY NA COMMODORE 64/128

- Niskie ceny.
- Co piąty program darmo! (Przy zamówieniu powyżej 20 pozycji rabat 25%). ltd.
- Katalog gratis.

Informacje: koperta + znaczek.
PROJECT X (Jacek K.)
ul.Kilińskiego23E/4, 59-300 Lublin.

KLUB KOMPUTEROWY „AMIGA"
97-400 BEŁCHATÓW
ul. Kątna 46
Przyjmujemy zapisy osób z całej Polski.
Dla członków klubu proponujemy ponad 2000 programów komputerowych po najniższych cenach w Polsce (2000zł). Informacje - koperta+znaczek.

C-64, magnetofon, 2 joysticki, cartridge, kasety-tanio sprzedam. Sosnowiec tel. 63-59-17.

Kupię dokumentację i literaturę do C-64. Lenart L. skr.87, 48-100 Głubczyce.

OPROGRAMOWANIE GEODEZYJNE na C-64 kupię. Janusz Hoffman, 14-300 MORĄG, ul. Pułaskiego 9/10.

Klub Komputerowy Stodoła
AMIGA

- oferuje najlepsze stacje dysków 3,5" i 5,25"
- serwis sprzętu firmy Commodore
- literatura (także 64 plus 4)
- akcesoria itp.

Zapraszamy codziennie, oprócz sobót i niedziel w godzinach 11⁰⁰ - 20⁰⁰
Warszawa ul. Batorego 10
tel. 25-60-31 wew. 35.
Giełdy komputerowe w Stodole, sobota od 10⁰⁰ - 15⁰⁰

64 PLUS 4

miesięcznik nr 5(7) maj 1991
cena 1 egz.: 5000 zł



Wydawca:
ABUK Spółka z o. o.

Redakcja nie ponosi odpowiedzialności za treść ogłoszeń.

Adres redakcji: Redakcja „64 plus 4" 85-166 Bydgoszcz 43 skrytka pocztowa 64
Redagują: Marcin Dudar, Samborkuźma, Paweł Sołtysiński, Waldemar Szczygieł (red. nacz.), Robert Turliński, Wojciech Wasilewski.
Okładka: Sławomir Karolczak.
Skład: ABUK
Druk: Z.P. POLRASTER 85-353 Bydgoszcz, ul. Orawska 19

Drodzy Czytelnicy!

Na drodze rozwoju naszego pisma uczyniliśmy kolejny krok - zwiększyliśmy objętość numeru do 28 stron (bez zmiany ceny!). Przewidujemy dalsze powiększanie ilości stron.

Nowa szata graficzna, poprawa jakości papieru spotkała się z Waszym uznaniem. Przy okazji - dziękujemy za uwagi i sugestie dotyczące wyglądu i tematyki „64 PLUS4".

Na życzenie użytkowników komputerów C-16 przerwaliśmy w tym numerze cykl „Mapa pamięci", zamieszczając artykuł „Grzebanie w bajtach"- w którym znajdziecie szereg ciekawych informacji praktycznych, bardzo przydatnych przy pisaniu własnych programów. Końcową część mapy pamięci zamieścimy za miesiąc.

W ostatnim okresie czasu nastąpił znaczny spadek cen sprzętu komputerowego, coraz szerszą grupę ludzi stać na jego zakup. Bardzo nas cieszy stale rosnąca liczba commodorowców. Pamiętajcie, aby tym „nowym" polecić lekturę naszego pisma!

Zbliża się koniec roku szkolnego... Mamy nadzieję, że nikomu z naszych czytelników chodzących do szkoły nie zdarzy się „wegetowanie" w tej samej klasie. Jeśli macie wątpliwości to odstawcie swój ukochany komputer - niech trochę odpocznie, w wakacje będzie sporo czasu na zabawę.

W wakacyjnych numerach naszego miesięcznika postaramy się zamieścić więcej „lżejszych" materiałów.

Do zobaczenia za miesiąc!

Redakcja

Z daleka i z bliska

- Pierwszy polski magazyn dyskowy „Kebab" zakończył swój żywot. Ostatni piąty numer ukazał się w lutym br.

- Jeden z najlepszych polskich koderów: Mac/Katharsis pracuje nad nową wersją Virus Experta. Będzie ona nosiła numerek 2.0 i będzie produktem komercyjnym.

- Uwaga na programy łamane przez SKID ROW! Ostatnio wypuszczone „crack'i" wcale nie są 100% okay, jak to szumnie informują czołówki. Gry się wieszają, brak niektórych sekwencji. Złośliwi proponują zmianę nazwy na SHIT THROW.

- W Szczecinie odbyła się pierwsza w historii wojna pomiędzy Atarowcami i Spektrymowcami. Oba zespoły biły się do ostatniego bita. Oba poniosły ciężkie straty. Ulubioną formą ataku był śrubokręt wsadzony w szynę komputera. Efektem batalii jest spadek pogłowia Atari i Spektrum. Plotki mówią, że całą wojnę zaczął przebrany za Atarowca człowiek z klanu Commodore.

- Znowu nowy wirus! Tym razem dotarł do nas prosto z Islandii. Nosi dziwną nazwę „CCCP" ! Czyżby był z tamtejszy ?! Pięknie się linkuje!

- Uwaga, uwaga - wirusy znowu atakują. Większość niemieckich BBS'ów jest zarażona nowym bardzo złośliwym wirusem działającym z twardym dyskiem.

- Historyczna już grupa Wild Copper żyje! Pracują obecnie nad grą. Należy spodziewać się HITU !

- Na świecie sprzedano już 2.000.000 (2 miliony) Amig. Ciekawe ile jest ich w Polsce ?

- Grupa Anarchy uruchomiła swój pierwszy BBS. Nazywa się on „Provication" i można się z nim łączyć pod numerem +46 392 22084.

- O tym, że Amiga jest najlepszym komputerem na świecie wszyscy nie wiedzą. Stanowczo przeczą temu zwłaszcza właściciele innych komputerów. Środowisko Amigowców podjęło ostatnio szeroko zakrojoną akcję nawracania na prawidłową wiarę. Oto dialog który odbył się w sklepie Apple.

Pytanie: Ile kolorów może wyświetlić MacIntosh ?
Odpowiedź: 16,7 miliona
P: Wiem, taka jest paleta, ale ile może wyświetlić na raz ?
O: 16,7 miliona.
P: 16,7 miliona w tym samym czasie?
O: Tak
P: W jakiej rozdzielczości ?
O: 1024 na 768
P: Hmm... Przy tej rozdzielczości na ekranie będzie 786432 punktów. Jak można wyświetlić 16,7 miliona kolorów na 786432 punktach ?
O: Taaak... to było tylko teoretycznie
P: Więc po raz czwarty: ile kolorów może wyświetlić Mac ?
O: 256...

Ciekawostki zbierał:

HI - Man

IV Ogólnopolski Zlot Użytkowników Komputerów Commodore Tychy 1991

W dniach 6 i 7 kwietnia bieżącego roku w Spółdzielczym Domu Kultury „Tęcza” w Tychach odbył się IV Ogólnopolski Zlot Użytkowników Komputerów Commodore.

Organizatorzy wysłali ponad dwa tysiące imiennych zaproszeń do różnych osób i instytucji. Nasza redakcja również otrzymała takie zaproszenie - dziękujemy.

Na zlot przybyło ponad trzysta osób z całego kraju.

Karta wstępu kosztowała 25.000 zł. Za sumę tą organizatorzy zapewniali stolik, źródło prądu (również wszelkiego rodzaju przedłużacze, trójniki itp.), butelkę napoju, kanapki, okolicznościowe nalepki i znaczki, również możliwość rezerwacji noclegu.

Nad sprawnym przebiegiem imprezy czuwała grupa sympatycznych, elegancko ubranych, młodych ludzi - głównie tyskich fanów komputerów firmy Commodore.

Wszystkich uczestników zlotu obowiązywała stała, rygorystycznie przestrzegana, zasada wzajemnej wymianu oprogramowania - kto chce handlować - musi opuścić salę!

W naszej, nieuporządkowanej w zakresie praw autorskich rzeczywistości, przy braku legalnej i dobrze zorganizowanej dystrybucji oprogramowania, forma bezpłatnej wymiany jest - mimo zastrzeżeń - tzw. mniejszym złem niż niczym nie skrępowane zarabianie na sprzedaży cudzej własności.

Firma TEST z Katowic wszystkim pechowcom, którym w czasie trwania zlotu przytrafiły się awarie sprzętu, świadczyła szybko i solidnie usługi serwisowe, pobierając opłatę tylko za elementy elektroniczne.

Nad sprawnością sieci zasilających czuwało dwóch elektryków, gotowych w każdej chwili „dorzucić” kilka kilowatów.

Kulminacyjnym punktem pierwszego dnia zlotu było losowanie atrakcyjnych nagród specjalnych (min. stacja 1541) wśród wszystkich uczestników.
Fundatorami nagród byli: firma TEST z Katowic, firma BROWN ELECTRONIC, oddział tyski firmy POLFROST oraz URZĄD MIEJSKI w Tychach.

Instytucje te sponsorują organizatorów zlotu - Towarzystwo Użytkowników Komputerów działające przy SDK TĘCZA w Tychach. Gospodarze z uznaniem wyrażali się o swych sponsorach - twierdząc, że bez ich pomocy działalność towarzystwa była by niemożliwa.

Wymieniając osoby przychylne tej działalności nie sposób pominąć kierownictwa SDK TĘCZA, które nie tylko udostępnia swoje pomieszczenia, ale aktywnie wspiera wszelkie działania na rzecz krzewienia informatyki wśród młodzieży.

W tak trudnych dla kultury czasach (brak środków finansowych, wysokie koszty uyrzymania budynków, płace itp.) przykład Tych dowodzi, że dla chcącego ...

Towarzystwo Użytkowników Komputerów powstało w 1985r. Założycielami - i do dziś jego filarami - są Grzegorz Stolecki, Piotr Mięsowicz i szef - Krzysztof Grochalski.

Ich celem jest krzewienie informatyki, integracja środowiska, wzajemna pomoc. Nie bez znaczenia są efekty wychowawcze uzyskiwane w pracy z młodzieżą.

Pierwsz zlot odbył się w 1988r. Podobne spotkania odbywały się już w Krakowie i Opolu. Tychy leżą dokładnie w połowie drogi między tymi miastami - a może by tak spotkać się w środku? I tak to się zaczęło ...

Tegoroczne spotkanie było również okazją do szerszej dyskusji na temat naszego pisma.

W niedzielne popołudnie zmęczeni organizatorzy i zadowoleni uczestnicy żegnali się: do zobaczenia za rok!

Waldemar Szczygieł

Grzebanie w bajtach

Wsród użytkowników wszystkich chyba typów komputerów domowych niezwykle popularne są wszelkiego rodzaju „tajemnicze" instrukcje POKE I PEEK umożliwiające wykorzystanie możliwości komputera stardandowo niedostępnych w języku BASIC. Te „sztuczki" opierają się na ogół na manipulacji zawartością komórek pamięci przechowujących ważne dla systemu operacyjnego bądź interpretera BASIC'a wartości - np. adresy procedur, do których system odwołuje się, aby wykonać określone operacje (komórki przechowujące takie adresy nazywamy wektorami), adres aktualnie wykonywanej linii programu itp. Niektóre z tych komórek sprzężone są z układami wejścia/wyjścia i umożliwiają bezpośrednie sterowanie urządzeniami zewnętrznymi - np. monitorem czy magnetofonem. Wreszcie można odwołać się rozkazem SYS bezpośrednio do konkretnych procedur systemu lub interpretera BASIC'a zawartych w pamięci ROM. Poniżej przedstawiam zbiór użytecznych „sztuczek" tego typu dla komputerów Commodore 16/116/+4.

Komórki pamięci o adresach 43-56 zawierają adresy rozmaitych obszarów pamięci wykorzystywanych przez interpreter BASIC'a, między innymi:

- komórki 43-44 zawierają adresy początku programu w BASIC'u (w ogólnie przyjętym dla procesora 6502 formacie zapisu liczb dwubajtowych: najpierw młodszy bajt, potem starszy bajt - czyli adres otrzymujemy dodając do zawartości pierwszej z tych komórek zawartość drugiej pomnożoną przez 256). Adres ten standardowo wynosi $1001 (dziesiętnie 4997) lub, w przypadku komputera z pamięcią 64 KB i używania grafiki wysokorozdzielczej, $4001 (dziesiętnie 16385).
- komórki 55-56 zawierają adres końca pamięci używanej przez interpreter BASIC'a. Standardowo wartość ta ustawiona jest na koniec dostępnego obszaru pamięci RAM.

Zmieniając zawartości tych komórek możemy tworzyć wolne obszary pamięci, które nie będą wykorzystywane przez interpreter BASIC'a, zatem możemy swobodnie umieszczać tam własne dane, np. podprogramy w kodzie maszynowym, zdefiniowane własne tablice znaków itp. Tak więc:

- powiększając adres zapisany w komórkach 43-44 (zmniejszyć go nie można, gdyż poniżej znajduje się obszar pamięci ekranu!) uzyskamy wolny obszar pamięci pomiędzy adresem $1001 (lub $4001) a zdefiniowanym początkiem programu (należy pamiętać, że w komórce położonej bezpośrednio przed początkiem programu musi koniecznie znajdować się wartość zero! - jest to konieczne dla poprawnego działania interpretera BASIC'a). Oczywistym jest, że zmiany zawartości tych komórek można dokonać tylko wtedy, gdy w pamięci nie ma programu w BASIC'u. Po dokonaniu zmiany zawartości tych komórek należy przed rozpoczęciem wpisywania lub ładowania programu wykonać instrukcję NEW dla poprawnego ustawienia pozostałych wskaźników interpretera BASIC'a.
- Zmniejszając adres zapisany w komórkach 55-56 (jest to metoda najczęściej stosowana, gdyż można jej używać wewnątrz programu w BASIC'u) rezerwujemy obszar pamięci pomiędzy zdefiniowanym końcem pamięci dla BASIC'a a faktycznym końcem pamięci RAM. Po dokonaniu zmiany zawartości komórek 55-56 należy użyć instrukcji CLR.

W programie korzystającym z danych zawartych w instrukcjach DATA przydatna może być informacja, z której lini przeczytana została aktualna dana (np. dla wypisania komunikatu „Nieprawidłowa dana w linii nr ..."). Numer ten odczytać można z komórek 63-64, np. instrukcją:

```
NR = PEEK(63) + 256 * PEEK(64)
```

Komórki pamięci o adresach 774-775 zawierają tzw. wektor LIST, czyli adres procedury, do której interpreter BASIC'a odwołuje się każdorazowo przy listowaniu treści linii programu. Standardowym adresem tej procedury jest $8B6E. Jeżeli zmienimy ten wektor tak, aby wskazywał adres procedury „pustej" (nie wykonującej żadnych czynności), program nie będzie listowany - przy listowaniu pojawiać się będą jedynie numery linii. Adresem takim może być np. $CFBE, co uzyskujemy przez wpisanie:

```
POKE 774,190 : POKE 775,207
```

Standardowy adres procedury LIST - $8B6E - przywracamy rozkazami:

```
POKE 774,110 : POKE 775,139
```

Trudniej wytłumaczalne, nawet przy dobrej znajomości systemu, jest działanie instrukcji:

```
POKE 22,35
```

która powoduje pomijanie przy listowaniu programu numerów linii. Stan normalny przywraca się przez:

```
POKE 22,25
```

Komórki 816-817 zawierają wektor procedury SAVE, wykorzystywanej przez system operacyjny do zapisu programów na taśmę lub dysk. Instrukcje:

```
POKE 816,38 : POKE 817,242
```

spowodują ignorowanie polecenia zapisania programu. Jeżeli zamiast 38 do komórki 816 wpiszemy zawartość 39, dodatkowo podczas próby wykonania instrukcji SAVE w BASIC'u będzie sygnalizowany błąd. Brutalniejszą reakcję na próbę zapisania programu możemy uzyskać poprzez wpisanie do wektora SAVE adresu $FFF6:

```
POKE 816,246 : POKE 817,255
```

W tym przypadku próba zapisania programu spowoduje natychmiastowe wyresetowanie komputera.

Do standardowego adresu procedury SAVE - $F1A4 powracamy przez:

```
POKE 816,164 : 817,241
```

Komórki 806-807 zawierają tzw. wektor STOP. Procedurę tę system operacyjny, jak i interpreter BASIC'a, wywołuje podczas prawie każdej wykonywanej operacji, sprawdzając czy w czasie jej trwania nie został wciśnięty klawisz RUN/STOP. Standardową wartością tego wektora jest $F265. Zmiana tej wartości na inną instrukcjami POKE wymaga pewnej ostrożności, gdyż interpreter BASIC'a odwoła się do tej procedury także pomiędzy wykonaniem jednej i drugiej instrukcji POKE! Wartości podane w instrukcji POKE muszą być zatem tak dobrane, aby w tym czasie nie nastąpiło „zawieszenie się" komputera. Jeżeli zatem wykonamy instrukcje:

```
POKE 807,228 : POKE 806,76
```

(koniecznie w takiej kolejności!), system operacyjny komputera nie będzie sprawdzał klawisza RUN/STOP, a przerwanie tym klawiszem jakiejkolwiek wykonywanej operacji będzie niemożliwe (za wyjątkiem operacji wejścia/wyjścia z magnetofonem, w których nie można zablokować klawisza RUN/STOP, gdyż system sprawdza go poprzez bezpośredni odczyt klawiatury, a nie za pośrednictwem procedury STOP).

Wektor STOP można również wykorzystać do „ogłupienia" naszego komputera, wykorzystując fakt, iż jest

on wywoływany podczas prawie każdej operacji. Wykonajmy instrukcje:

```
POKE 806,28 : POKE 807,128
```

Komputer wyresetuje się, ale od tej pory będzie się resetował przy próbie wykonania jakiejkolwiek instrukcji, aż do momentu gdy wyresetujemy go „na dobre" wciśnięciem klawisza RESET lub równoważną instrukcją SYS 65526.

Jeżeli z kolei wykonamy instrukcję:

```
POKE 807,146
```

(zakładając, że wektor STOP miał wcześniej standardową wartość $F265), to komputer „zgłupieje" w inny sposób - najczęściej „OUT OF DATA ERROR", ale pojawiają się także inne efekty. Z uwagi na wspomniany specyficzny charakter wektora STOP nie polecam przywracania mu wartości standardowej za pomocą instrukcji POKE. Zamiast tego proponuję wykorzystać procedurę systemu operacyjnego, która ustawia na standardowe wartości wszystkie wektory systemu w komórkach 786-817 (w tym m.in. wektory SAVE i STOP):

```
SYS 65418
```

Natomiast wektory interpretera BASIC'a w komórkach 768-785 (m.in. wektor LIST) można ustawić na standardowe wartości procedurą:

```
SYS 33047
```

Komórka pamięci o adresie 198 zawiera kod aktualnie wciśniętego klawisza (kody klawiszy zawierają się w zakresie 0-63 i nie równają się kodom ASCII znaków). Wartość 64 w tej komórce oznacza, że żaden klawisz nie jest wciśnięty. Instrukcja:

```
WAIT 198,63
```

czeka na wciśnięcie dowolnego klawisza, a instrukcja:

```
WAIT 198,64
```

czeka na puszczenie wciśniętego właśnie klawisza.

Komórka 1347 zawiera natomiast stan klawiszy specjalnych: SHIFT, CTRL i C=, których wciśnięcie nie rejestruje się w komórce 198. Poszczególne bity tej komórki mają następujące znaczenie:

bit 0 ustawiony, gdy wciśnięty klawisz	SHIFT	(wartość 1)
bit 1 „	C=	(wartość 2)
bit 2 „	CTRL	(wartość 4)

Zatem np. gdy jednocześnie wciśnięte są klawisze CTRL i SHIFT zawartością komórki 1347 będzie 5 (1+4).

Kody ASCII znaków wprowadzanych z klawiatury zapisywane są w buforze klawiatury znajdującym się w pamięci pod adresami 1319-1328 i stamtąd pobierane są przez instrukcje takie jak INPUT czy GET. Komórka o adresie 239 zawiera liczbę znajdujących się aktualnie w buforze znaków. Jeżeli chcemy „zlikwidować" znaki w buforze, np. po to, aby najbliższa instrukcja GET nie odczytała, zamiast znaków wprowadzonych przez użytkownika z klawiatury, „starych" znaków, wciśniętych uprzednio (np. podczas długiego obliczania) i dotąd nie przetworzonych, możemy to uzyskać instrukcją:

```
POKE 239,0
```

Wpisując do komórki 239 inne liczby, włącznie z wpisywaniem odpowiednich wartości do bufora klawiatury, możemy „udawać" wprowadzenie określonych znaków z klawiatury, np. po to, aby w programie w BASIC'u skorzystać z pewnych funkcji monitora języka maszynowego. Ta technika programowania (zwana „dynamiczną" lub „symulowaną" klawiaturą) wymagałaby jednak nieco dłuższego i bardziej szczegółowego omówienia. Pozostawiam ją dociekliwym, a jeżeli czas pozwoli postaram się ją opisać osobno.

Skoro już o klawiaturze mowa, jeszcze jedną „sztuczkę" dotyczącą klawiatury umożliwia nam komórka o adresie 1344. Normalnie każdy klawisz na klawiaturze przyciśnięty przez dłuższy czas jest automatycznie powtarzany. Po wykonaniu POKE 1344,64 efekt powtarzania jest wyłączony, a po POKE 1344,0 - efekt występuje tylko dla klawiszy kursora, DEL i spacji. Powrót do normalnego stanu przez POKE 1344,128.

Następująca sekwencja instrukcji pozwala ustawić kursor w żądanym miejscu ekranu:

```
POKE 205, wiersz : POKE 202, kolumna : SYS 55363
```

Kolejna grupa „sztuczek" wiąże się z operowaniem rejestrami sterownika graficznego TED. I tak wykonując instrukcję:

```
POKE 65286, PEEK(65286) OR 16
```

włącza ekran ponownie.

Wykonanie instrukcji:

```
POKE 65287, PEEK(65287) OR 128
```

daje możliwość wyświetlenia na ekranie na raz obu zestawów znaków (normalnie przełączanych wciśnięciem klawiszy C= i SHIFT); znaki z drugiego zestawu są wyświetlane zamiast negatywowych obrazów odpowiednich znaków z pierwszego zestawu. Powrót do sytuacji normalnej (jeden zestaw znaków + negatywy) następuje po:

```
POKE 65287, PEEK(65287) AND 127
```

Z kolei instrukcja:

```
POKE 65286, PEEK(65286) OR 64
```

redukuje liczbę znaków możliwych do wyświetlenia do połowy zestawu (znaki o kodach ekranu 0-63), ale za to możemy mieć na ekranie aż cztery kolory tła! Znaki wciskane z klawiszem SHIFT będą wypisywane na tle dodatkowym nr 1, zamiast znaków w negatywie otrzymamy tło dodatkowe nr 2, a zamiast znaków w negatywie i z klawiszem SHIFT ukażą się znaki na tle dodatkowym nr 3. Dodatkowe kolory tła określane są przez zawartości następujących komórek:

tło dodatkowe 1 - komórka 65302
tło dodatkowe 2 - komórka 65303
tło dodatkowe 3 - komórka 65304

Do komórek tych należy wpisać wartości równe (nr koloru - 1) + 16 * jaskrawość.

UWAGA: W tym trybie pracy sterownika nie widać kursora (nie działa także kod sterujący migotaniem znaków), dlatego też polecam jego użycie tylko wewnątrz programów. Powrót do normalnego trybu przez:

```
POKE 65286, PEEK(65286) AND 191
```

Przy okazji wspomnieć można, że komórka 65301 przechowuje zapisaną w analogiczny sposób barwę „normalnego" tła obrazu, a komórka 65305 - barwę ramki ekranu. Można je wykorzystać np. do nadania ramce koloru identycznego z tłem:

```
POKE 65305, PEEK(65301)
```

(co jest odpowiednikiem instrukcji COLOR 4, RCLR(0), RLUM(0)). Komórka o adresie 1339 przechowuje bieżący kolor, jakim pisany będzie tekst (ustawiany instrukcją COLOR 1, ...). W tym przypadku do koloru zapisanego w powyższy sposób dodać można jeszcze 128 dla atrybutu migotania znaków. Aby ustawić kolor znaków identyczny z kolorem tła (drukowany tekst będzie niewidoczny), możemy posłużyć się instrukcją:

```
POKE 1339, PEEK(65301)
```

(odpowiednikiem BASIC'owym tej instrukcji jest COLOR 1, RCLR(0), RLUMC(0)).

Sterownik TED daje nam możliwość definiowania własnych wzorców znaków w pamięci RAM (np. zawierających polskie litery) i użycia ich zamiast standardowych. Zasady definiowania własnych znaków wymaga oddzielnego omówienia; tu chciałbym podać tylko instrukcje powodujące uaktywnienie własnej tablicy znaków w pamięci RAM:

```
POKE 65298, PEEK(65298) AND 251 : POKE 65299, adres
```

gdzie „adres" jest starszym bajtem adresu, pod jakim umieściliśmy naszą tablicę znaków w pamięci RAM. Tablica znaków może być umieszczona tylko pod adresami stanowiącymi wielokrotność $400, czyli np. (12288), $3400 (13312), $3800 (14336), $3C00 (15360) itp. Odpowiednimi wartościami do wpisania w komórkę 65299 byłyby $30=48, $34=52 etc.

Standardowa zawartość komórki 65298 wynosi 196, i jeżeli nie pracują równocześnie generatory dźwięku (również wykorzystujące tę komórkę), powyższe instrukcje można uprościć do postaci:

```
POKE 65298, 192 : POKE 65299, adres
```

Standardową tablicę znaków włączamy ponownie instrukcjami:

```
POKE 65298, PEEK(65298) OR 4 : POKE 65299,208
```

albo prościej:

```
POKE 65298,196 : POKE 65299,208
```

UWAGA: Gdy mamy aktywną własną tablicę znaków w RAM, po każdym komunikacie błędu w BASIC'u, wyjściu z monitora języka maszynowego do BASIC'u lub wyjściu z trybu graficznego do tekstowego (GRAPHIC 0) zawartość komórki 65298 „psuje się" w ten sposób, że na ekranie pojawia się nieczytelna „kasza". W takiej sytuacji, aby przywrócić normalny obraz na monitorze, należy wykonać instrukcję (niestety, trzeba ją wpisać „na ślepo"):

```
POKE 65298,192
```

Powyższe uwagi dotyczą oczywiście znaków wyświetlanych w trybie tekstowym. Aby zmienić tablicę znaków wyświetlanych w trybie graficznym przez instrukcję CHAR, należy wykonać instrukcję:

```
POKE 740, adres
```

gdzie „adres" ma takie samo znaczenie jak powyżej. Do standardowej tablicy znaków wracamy poprzez POKE 740,208. Natomiast aby „zmusić" instrukcję CHAR do posługiwania się znakami drugiego zestawu (zawierającego małe litery) trzeba użyć POKE 740,212.

Osobom piszącym długie programy korzystające z grafiki wysokorozdzielczej przeszkadza zapewne fakt, że przy pierwszym wykonaniu instrukcji GRAPHIC na ekranie przez dosyć długi czas widoczne są „śmieci". Dzieje się tak dlatego, że program jest „przenoszony" spod adresu $1001 pod adres $4001, co wymaga nieco czasu, szczególnie jeżeli program jest długi. Dla uniknięcia pojawiania się owych „śmieci" na ekranie można, zanim wykonana zostanie jakakolwiek instrukcja GRAPHIC, wcześniej przetransportować program pod adres $4001 rozkazem:

```
SYS 50748
```

Pisząc programy składające się z kilku kolejno ładujących się części (nakładek) stajemy nieraz przed problemem: jak pierwsza część programu ma rozpoznać, czy została załadowana z taśmy czy dysku i w zależności od tego podjąć ładowanie dalszych części z odpowiedniego urządzenia. Otóż numer ostatnio wykorzystywanego urządzenia zewnętrznego znajduje się w komórce o adresie 174. Należy więc użyć np. sekwencji rozkazów:

```
U=PEEK(174) : LOAD „część_2",U
```

Nie można (!) natomiast użyć instrukcji LOAD „część_2", PEEK(174), gdyż przed rozpoczęciem analizy parametrów instrukcji LOAD interpreter BASIC'a wstępnie ustawia zawartość komórki 174 na 1, tak więc taka instrukcja podejmowałaby ładowanie zawsze z magnetofonu.

Po zakończeniu operacji ładowania z taśmy czy dyskietki, końcowy adres załadowanego bloku powiększony o 1 znajduje się w komórkach $9D-9E (dziesiętnie 157-158).

Przy ładowaniu z magnetofonu jego bufor, znajdujący się w pamięci pod adresami $332-3F2 (dziesiętnie 818-1010), przechowuje następujące dane:

$333-334 - adres początku ładowanego bloku

$335-336 - adres końca ładowanego bloku + 1

od $337 - nazwa programu (e w.dalsza część nagłówka, jeżeli program był zapisany w którymś z systemów TURBO)

Dla użytkowników stacji dysków przydatne mogą być dwie poniższe „sztuczki" dotyczące odczytywania kanału błędu stacji dysków poprzez zmienną DS$. Zmienna ta standardowo dotyczy stacji dysków nr 8; jeżeli posiadamy więcej niż jedną stację dysków i chcemy odwoływać się także do innych stacji, możemy to uczynić wykonując przed odwołaniem do zmiennej DS lub DS$ instrukcję:

```
POKE 631, numer urządzenia
```

Odwołanie do zmiennej DS lub DS$ odczytuje kanał błędu stacji dysków tylko wtedy, gdy w komórce 121 jest wartość zero; w przeciwnym razie zwraca zapamiętaną ostatnią wartość tej zmiennej. Normalnie, wszystkie instrukcje wejścia/wyjścia takie jak LOAD, SAVE, OPEN itp. zerują komórkę 121; jeżeli jednak obsługiwaliśmy operacje dyskowe niestandardowo, np. własnymi procedurami w kodzie maszynowym wywoływanymi przez SYS, lub chcemy kolejno odczytać kanały błędów kilku stacji dysków posługując się „sztuczką" podaną powyżej, pamiętajmy przed odwołaniem do zmiennej DS$ o:

```
POKE 121,0
```

Na koniec wreszcie chciałbym podać sposób zapisywania bądź ładowania z taśmy/dysku dowolnych obszarów pamięci z programu w BASIC'u. Słabość instrukcji SAVE i LOAD Commodore BASIC'u jest tu bardzo widoczna. Ładowanie np. kodu maszynowego instrukcją LOAD powoduje po jej wykonaniu restart programu od początku; instrukcją SAVE zapisać można tylko program w BASIC'u, zapis innych obszarów pamięci wymaga użycia monitora języka maszynowego, a nawet monitor nie potrafi wczytać programu z relokacją, tzn. pod inny adres niż ten spod którego został zapisany. Wszystkie te niedogodności usuwa następujące postępowanie:

```
SAVE:
     SYS 43115 „nazwa" [,urządzenie(,adres sekundarny)]
     POKE 178,mł.bajt : POKE 179, st.bajt (adres początku
                              obszaru do zapisania)
     POKE 157, mł.bajt : POKE 158, st.bajt (adres końca
                              obszaru do zapisania + 1)
     SYS 61857
LOAD:
     SYS 43115 „nazwa" [,urządzenie (,adres sekundarny)]
     [POKE 180, mł.bajt : POKE 161, st.bajt ](adres ładowania
     jeżeli relokacja czyli gdy adres sekundamy = 0)
     POKE 2034,0 : SYS 61511
```

Znaczenie parametrów w instrukcji SYS 43115 jest takie samo jak w odpowiednich instrukcjach SAVE bądź LOAD.

I na koniec tych wszystkich „sztuczek" niespodzianka dla dociekliwych pozostawiona w pamięci ROM przez twórców systemu operacyjnego i interpretera BASIC'a Commodore 16/116/+4: proszę wywołać SYS 52651 !

Jarosław Rafa

SŁOWNIK SLANGU

Cześć ! Dzisiaj kolejna porcja haseł do przyswojenia. Ostatnio poznawaliśmy zwroty zaczynające się na literę „A", dzisiaj wykonamy GOTO „B".

Background - tło np. w grach; background color - kolor tła.

Bam- wcale nie chodzi tu o wybuch lecz o specyficzne miejsce na dysku, w którym komputer zapisuje dane o rozmieszczeniu zbiorów.

Bank - jest to miejsce w pamięci RAM, w którym jest pełno $! Wszyscy właściciele C-64 mają aż cztery banki, a w każdym z nich po 16384 dolarów !

Bary - to kolorowe paski na ekranie, stojące lub poruszające się po różnych śmiesznych krzywych.

Bit - najmniejsza jednostka informacji mogąca przyjmować wartośc 0 lub 1 (patrz: - bity mu pogasły; - zgaś bit).

Block - miejsce na dyskietce, w którym komputer zapisuje dane.

Bobs - graficzne „sprite'y", najczęściej w postaci kulki, które poruszając się po ekranie kreślą różne ciekawe krzywe lub układają się w rysunki (patrz: - vector bobs).

Boot - miejsce na dyskietce w którym jest zapisany program „wykonujący się" przy starcie systemu (patrz: - zabutuj dysk).

Border - miejcse na ekranie na którym komputer nie może wyświetlać grafiki. Ulubione miejsce koderów na umieszczanie grafiki w demkach.

Break - nazwa rozkazu BRK przerywającego wykonywanie programu; klawisz czyniący to samo co rozkaz BRK.

SUPLEMENT

Bity mu pogasły - termin oznaczający człowieka notorycznie kłamiącego lub niespełna rozumu.

Zgaś bit - ustaw bit na „0".

Vector bobs - Bobsy liczone matematycznie (wektorowo).

Zabutuj dysk - zresetuj system komputera i włóż do stacji daną dyskietkę.

Hi-Man

COMMODORE 116 (ULEPSZONA KLAWIATURA) z magnetofonem + 7 kaset z grami + literatura - tanio sprzedam. Marcin Woch, ul. Piaski 1a, 32-080 Zabierzów.

C-64 z magnetofonem. Poszukuję gier strategicznych. Piotr Nowotarski, 32-546 Miłoszowa 702.

Wymienię oprogramowanie, odkupię literaturę o Commodore 64. Artur Karaźniewicz, 11-111 Kraszewo 33, woj. olsztyńskie.





Assembler 6510

Lekcja 2

Miesiąc temu poznaliście systemy liczbowe z jakimi będziecie się stykać podczas programowania w języku maszynowym. Dziś pora poznać serce naszego C-64 - mikroprocesor 6510.

Jak to się dzieje, że komputer potrafi wykonywać operacje dodawania, mnożenia, drukować teksty i realizować wszystkie inne efekty, które oglądacie w programach. Wszystko to jest zasługą procesora 6510. Oczywiście potrzebne są również inne specjalizowane układy (SID, VIC, porty), ale one są „ożywiane" właśnie przez mikroprocesor.

Aby móc wykonywać wszystkie powierzone mu funkcje, 6510 (podobnie jak i inne procesory) posiada własną pamięć niezależną od RAM'u, którą nazywa się rejestrami. Nasz mikroprocesor jest dość ubogi w porównaniu do innych, bowiem ma ich tylko 6.

Po wczytaniu dowolnego monitora języka maszynowego na ekranie ukazuje się napis: (rejestry można również uzyskać naciskając „R")

```
PC   AC   XR   YR   SP   NV-BDIZC (lub SR)
```

Pod poszczególnymi literami podane są różne liczby. To co nam podaje monitor to właśnie spis zawartości rejestrów mikroprocesora. Oto ich opis:

PC to skrót od angielskich słów PROGRAM COUNTER czyli po polsku : Licznik Programu. Jest to rejestr szesnastobitowy (czyli zawierać może liczby od 0 do 65536). W zasadzie jest on o wiele ważniejszy dla samego mikroprocesora niż dla użytkownika. W rejestrze PC jest bowiem przechowywany adres wykonywanej instrukcji w pamięci. Bliżej jego działaniu przyjrzymy się potem.

AC to akumulator (ang. ACCUMULATOR). Jest to najważniejszy rejestr w naszym 6510. W nim są wykonywane wszystkie operacje dodawania, odejmowania, przesunięć itp. Jest to rejestr ośmiobitowy, czyli można w nim przechować liczby od 0 do 255.

XR i **YR** (X register i Y register) to rejestry X i Y. Są to tak zwane rejestry indeksowe ułatwiające adresowanie pamięci. Są również ośmiobitowe.

SP (Stack Pointer) to wskaźnik stosu. Pokazuje on pierwsze wolne miejsce na tzw. stosie. Co to jest stos, jak działa i jak go wykorzystywać dowiecie się później.

Ostatni z rejestrów jest różnie podawany przez różne monitory. W jednych jest napis **SR** (Status Register) a w innych cały rejestr jest rozpisany na bity. (NV-BDIZC). Czym jest rejestr znaczników ?

Jest to takie „miejsce" w którym procesor zapisuje stan w jakim się znajduje. Każdy z bitów ma swoje znaczenie:

N - znak (negativ)
V - nadmiar (overflow)
- - nieużywany
B - przerywanie (break)
D - tryb pracy BCD
I - maska przerwań (interrupt)
Z - zero
C - przeniesienie (carry)

Są one modyfikowane po większości operacji (np. dodawanie, przesunięcie itp.) i dają dodatkową informację o wyniku.

Uruchomcie wasz monitor języka maszynowego i wpiszcie sekwencję rozkazów:

```
1000 LDX #$00
1002 BRK
I 1000
```

Na ekranie ukażą się następujące cyfry:

```
PC AC XR VR SP NV — BDIZC
1002 00 00 00 F4 000    10010
```

W PC znajduje się liczba $1002 bo właśnie w tym adresie komputer napotkał BRK (przerwanie) i skończył wykonywać nasz program. Rozkaz LDX #$00 załadował do rejestru X wartość 0 (XR = 0). Spójrzcie jakie bity zapaliły się w rejestrze znaczników. Są to B i Z.

B - jak już wiecie - oznacza wykonanie przerwania programowego (Break - rozkaz BRK). W naszych przykładach będzie on się zawsze palił bowiem zawsze procedury będziemy kończyli break'iem.

Oprócz bitu B zapalił się także Z. Oznacza to, że wynik ostatniej operacji przeprowadzonej przez mikroprocesor równa się zero.

Jeśli teraz pierwszy rozkaz zmienicie na:

```
1000 LDX #$05
```

to XR = 05 a Z = 0 - wynik ostatniej operacji nie jest równy zeru.

Bardzo ważnym bitem jest C (Carry) - przeniesienie. Zapala się on wtedy gdy wynik ostatniej operacji nie mieści się na ośmiu bitach.

Wykonajmy następujący program:

```
1000 CLC
1001 LDA #$10
1003 ADC #$15
1005 BRK
```

Dodaje on do akumulatora (LDA #$10) wartość 15 (ADC #$15). A więc po wykonaniu operacji mamy w AC liczbę $25. Pamiętajcie jednak, że akumulator ma tylko osiem bitów (można więc w nim zapisać liczby od 0 do255). Co by się stało gdyby tak dodać dwie liczby, suma których była by większa od 255. Spróbujmy:

```
1000 CLC
```

```
1001  LDA  #$10
1003  ADC  #$F1
1005  BRK
```

W wyniku powinniśmy otrzymać wartość $ 101. A co mamy ? AC = 01, ale Carry (C) = 1 ! Nastąpiło tzw. przeniesienie. Wynik dodawania nie zmieścił się w ośmiu bitach.

Innym ważnym znacznikiem jest N. Otóż, jak już zauważyliście, komputer może przechowywać liczby tylko dodatnie. Czasami jednak przydatne jest zapisanie liczb ujemnych. Wprowadzono więc zasadę, że ostatni ósmy bit liczby oznacza znak (+ lub -). W ten sposób można zapisać cyfry od -128 do +127. Wygląda to następująco

```
00000001 = 01hex = 01dec = 1
8 . . . . . 1 (numer pozycji bitu)
11111111 = $FF = 255 = -1
```

Widzicie więc, że ta sama zawartość rejestru może oznaczać różne liczby.

Do sprawdzania znaku służy znacznik N. Jeśli się on pali to znaczy, że liczba jest ujemna (ostatni bit zapalony), a jeśli jest zgaszony to liczba jest dodatnia (ostatni bit zgaszony).

Wykonajmy:

```
1000  LDY  #$3E
1002  BRK
YR = 3E  N = 0
```

```
1000  LDY  #$FA
1002  BRK
YR = 85  N = 1
```

Zawartość rejestru Y możemy więc traktować jako $F5 lub $ -06!

Przeanalizujcie poniższe przykłady i spróbujcie napisać swoje. Znajomość zasad działania znaczników jest bardzo ważna.

```
1000  CLC
1001  LDA  #$F0
1003  ADC  #$0F
1005  BRK
AC = FF  N = 1  Z = 0  C = 0
1000  CLC
1001  LDA  #$F0
1003  ADC  #$10
1005  BRK
AC = 00  N = 0  Z = 1  C = 1
1000  CLC
1001  LDA  #$F0
1003  ADC  #$3E
1005  BRK
AC = 2E  N = 0  Z = 0  C = 1
```

Za miesiąc nauczymy się adresować pamięć i napiszemy prosty relokator.

Sambor Kuźma

JAK ZROBIĆ WŁASNE DEMO (2)

W poprzednim artykule z tego cyklu starałem się wyjaśnić zasady działania i obsługi przerwań programowych w Commodore 64. Umiejętność sprawnego pisania własnych procedur obsługujących przerwania leży u podstaw efektywnego i ...efektownego programowania Twojego komputera. Ci z Czytelników, którzy często przyglądają się możliwościom innych komputrów (a zwłaszcza oprogramowaniu zawartemu w ich pamięciach stałych) z łatwością stwierdzą, że pomimo słusznie podkreślanych dużych możliwości graficzno-dźwiękowych, Commodore 64 jest komputerem trudnym do opanowania dla początkującego programisty. Wystarczy przypomnieć sobie o dość dotkliwym braku instrukcji graficznych (nie wspominając o rozkazach do tworzenia muzyki). O ileż łatwiej mają choćby posiadacze małego Atari, gdzie łączenie grafiki z tekstem jest możliwe w wygodny sposób z poziomu interpretera języka BASIC. Najczęściej posiadacz C-64 rozwiązuje ten problem poprzez uruchomienie jednego z wielu dostępnych rozszerzeń BASIC, gdzie podziału ekranu na część grafi czną i tekstową dokonuje się za pomocą np. komendy TEXT (w Graphic Basic). Najczęściej nikt się nie zastanawia nad sposobem uzyskania takiego efektu, ale my chcielibyśmy to wiedzieć ! Nie będziemy tym razem zastanawiać się nad tym, jak napisać własną instrukcję PLOT czy DRAW, bo to temat na osobny artykuł - dziś postaram się wytłumaczyć, jak działa mechanizm podziału ekranu na dowolne obszary robocze.

Dla lepszego wyjaśnienia powróćmy do „małego" Atari: odpowiednia komenda graficzna bezpośrednio modyfikuje tzw. display-list. Czym jest ów display-list ? Otóż w dużym przybliżeniu, jest to lista rozkazów procesora graficznego, gdzie odpowiednio umieszczone dane informują wspomniany powyżej układ o tym, co należy wyświetlać i w jakiej linii ekranu. Teoretycznie może to wyglądać w ten sposób: w pierwszych 10 liniach znakowych ma być uruchomiony tryb tekstowy, przez następnych 6 linii - grafika a pozostałe linie - ponownie tekst. „No tak - powiecie - zakładając, że potrafimy opanować pisanie komend display-list, to i tak pozostaje jeszcze jeden mały problem: w Commodore 64 NIE MA czegoś takiego jak display-list !". I będziecie mieli rację! Nie pozostaje nam nic innego, jak ...zasymulowanie listy rozkazów procesora graficznego przy pomocy poznanych w artykule z poprzedniego numeru „64 plus 4" przerwań graficznych (przerwań rastra).

Na początek ustalmy sobie, jaki cel chcemy osiągnąć. Może taki: pierwsze 4 górne wiersze ekranu mają być wyświetlane w trybie tekstowym, następne 10 w graficznym trybie wysokiej rozdzielczości a od 14 linii do końca - ponownie tryb tekstowy. Ustalmy także, że tryb tekstowy będzie operował na normalnym obszarze ekranu tekstowego (a więc pod adresem $0400 w pierwszym banku graficznym), część graficzną zaś generować będzie obraz obszaru pamięci pod adresem $E000-$FF40 z ekranem atrybutów pod adresem $CC00. Aby dokonać tych „przełączeń" będziemy podczas obsługi przerwań zmieniać zawartości następujących komórek w układzie graficznym naszego Commodore 64:

$D011 - komórka odpowiedzialna m.in. za przełączanie pomiędzy trybem tekstowym i graficznym.

$D018 - komórka odpowiedzialna za położenie generatora znaków, położenie ekranu tekstowego, atrybutów oraz obszar BITMAP.

$DD00 - komórka odpowiedzialna m.in. za przełączenie banków graficznych.

Aby uruchomić ekran tekstowy, do komórek tych należy wpisać co następuje:

$1B do komórki $D011,

$15 do komórki $D018,

$97 do komórki $DD00,

...aby uruchomić tryb wysokiej rozdzielczości analogicznie należy wpisać:

$3B do komórki $D011

$3E do komórki $D018

$94 do komórki $DD00

Jeśli zastanowimy się nad założeniami naszego zadania, nie trudno zauważyć, że w zasadzie podczas wyświetlania obrazu przełączenie trybów wystarczy przeprowadzić tylko dwa razy - przy otwieraniu obszaru graficznego i przy jego zamykaniu. Pierwsze przełączenie musi nastąpić na wysokości 4 od góry linii tekstowej (włączamy tryb HIRES) a drugie na wysokości linii 14 (włączamy ponownie tryb TEXT). Wniosek: należy wywołać przerwanie graficzne w tych dwóch miejscach. Jak zapewne pamiętacie z poprzedniego artykułu, numer pierwszej linii ekranowej dla VIC'a jest równy $32, a każda linia tekstowa jest równa ośmiu liniom rastra. Tak więc numer linii rastrowej na wysokości 4 wiersza ekranu można obliczyć z tego prostego wzoru:

linia1 = $32+4 x 8 = $52

a linii na wysokości 14 wiersza:

linia2 = $32+14 x 8 = $A2

Posiadając te informacje możemy przystąpić do napisania procedury obsługującej taki podział ekranu:

```
ORG   $C000
:linia1   = $52
:linia2   = $A2
      SEI              ;definiowanie   przerwań   graficznych
      LDA   #$7F
      STA   $DC0D
      LDX   #$00
      STX   $DC0E
      INX
      STX   $D01A
      LDA   #$1B
      STA   $D011
      LDA   #linia1
      STA   $D012
      LDA   #<irq1
      STA   $0314
      LDA   #>irq1
      STA   $0315
      CLI
      RTS
:irq1 LDA   #$3B       ;otwieramy   ekran   HIRES
      STA   $D011
```

```
        LDA  #$3E
        STA  $D018
        LDA  #$94
        STA  $DD00
        LDA  #linia2
        STA  $D012
        LDA  #<irq2
        STA  $0314
        LDA  #>irq2
        STA  $0315
        INC  $D019
        JMP  $EA7E
:irq2   LDA  #$1B       ;otwieramy ekran TEXT
        STA  $D011
        LDA  #$15
        STA  $D018
        LDA  #$97
        STA  $DD00
        LDA  #linia1
        STA  $D012
        LDA  #<irq1
        STA  $0314
        LDA  #>irq1
        STA  $0315
        INC  $D019
        JMP  $EA31
```

Procedura ta została napisana przy użyciu Macroassemblera, ale myślę, że z powodzeniem może zostać uruchomiona pod innym assemblerem lub nawet napisana z wykorzystaniem dowolnego monitora pamięci.

Jak widać zastosowałem tutaj technikę dwukrotnego wywoływania przerwania graficznego podczas każdego wyświetlania obrazu. W tekście programu obsługi tych przerwań noszą one nazwy „irq1" i „irq2".

Paweł Sołtysiński

Bez komentarza

```
5 REM (W) PAWEL SOLTYSINSKI (64+4 & AMIGA)
10 AD=4096:X=0
20 READ C:IF C=0 THEN 40
30 POKE AD,C:X=X+C:AD=AD+1:GOTO 20
40 IF X12093 THEN PRINT „ZLE DANEI":STOP
50 PRINT:PRINT„WCISNIJ SPACJE...": PRINT
„SYS 4096 URUCHAMIA EFEKT PONOWNIE"
60 SYS 4096
70 :
100 DATA 120, 173, 18, 208, 201, 51, 208, 248, 165,
        162, 141, 33, 208, 120, 173, 18
101 DATA 208, 205, 18, 208, 240, 251, 238, 33, 208,
        234, 234, 238, 32, 208, 234, 234
102 DATA 238, 33, 208, 234, 234, 206, 33, 208, 234,
        234, 208, 32, 208, 234, 234, 206
103 DATA 33, 208, 173, 1, 220, 201, 255, 240, 2, 88,
        96, 200, 208, 208, 232, 208
104 DATA 205, 238, 33, 208, 238, 32, 208, 76, 14, 16, 0
```

Programy ciekawe, zwariowane i takie sobie

Najciekawsze programy zawsze znajduje się podczas robienia porządków na dyskietkach. Taka też jest historia tego programu znalezionego na zapomnianym dysku leżącym pod fotelem. Do tej pory nie wiem skąd ten dysk znalazł się pod fotelem, a tym bardziej kto nagrał na nim programik, który chcę wam przedstawić. Nosi on nazwę „INVERT SCREEN" i robi dokładnie to na co wskazuje jego nazwa. Nie wiem kto jest autorem ani gdzie był pierwszy raz publikowany dlatego przepraszam prawowitego właściciela za nie podanie jego nazwiska. Wersja, którą będziemy mogli wklepać do maszyny jest nieco krótsza od orginału. Kilka bajtów było zbędnych więc je usunąłem.

Program ten należy wprowadzić do komputera za pomocą dowolnego monitora języka maszynowego, a następnie zapisać go na dysku lub taśmie:

S"INVERT SCREEN", 08, 0801, 0889

(dla taśmy wartość 08 trzeba zmienić na 01 (normal) lub 07 (turbo final II/III)).

Uruchomienie następuje po wykonaniu RUN.

```
0800:00 0B 08 C7 07 9E 32 30
0808: 36 31 00 00 00 78 A9 33
0810: 85 01 A9 00 85 FB A9 DO
0818: 85 FC A0 00 B1 FB AE 03
0820: BF 4A 26 02 CA D0 FA 98
0828: 49 07 A8 A5 02 91 FB 98
0830: 49 07 A8 C8 D0 E6 E6 FC
0838: A5 FC C9 E0 D0 DE A2 00
0840: BD 65 08 9D 10 01 E8 E0
0848: 25 D0 F5 A9 37 85 01 A9
0850: 84 8D 18 D0 A9 00 8D 00
0858: DD A9 10 A2 01 8D 14 03
0860: 8E 15 03 58 60 A9 00 AA
0868: A8 88 BD 00 04 99 E8 E2
0870: BD 00 05 99 E8 E1 BD 00
0878: 06 99 E8 E0 BD 00 07 99
0880: E8 DF 88 E8 D0 E4 4C 31
0888: EA 00
```

Hi-Man & Polonus

RELOCATOR

Jedną z niezaprzeczalnych zalet pisania programów w kodzie maszynowym przy użyciu assemblera jest możliwość łatwej relokacji kodu wynikowego w dowolne, dogodne dla programisty miejsce w pamięci komputera. Zachowując przezornie nasze dzieło w postaci tekstu źródłowego, zawsze będziemy mogli do niego wrócić i ponownie wykorzystać w innym programie jako jego fragment. Wszystko to jest łatwe i wygodne, o ile posiadamy ...tekst źródłowy. Co bowiem można poradzić na gotowy już program, który chcielibyśmy mieć w pamięci komputera, tyle tylko, że w innym miejscu ?

Są na to dwa sposoby: użyć tzw. reassemblera lub przenieść go „ręcznie", np. korzystając z dowolnego monitora pamięci.

Obie metody mają swoje dobre i złe strony: pracujący reassembler czasami stara się być „za mądry" niszcząc np. teksty czy grafikę, gdyż umieszczone tam dane zostały rozpoznane jako fragmenty programu (tzn. jako rozkazy procesora) a następnie przerelokowane. Metoda „ręczna" jest przede wszystkim bardzo żmudna (jak sama nazwa wskazuje), ale za to pozwala uniknąć błędów, jakie mogą powstać przy użyciu programu reassemblującego. Na naszym PDP C-64 z marca znajduje się program, który powstał właśnie po to, aby do maksimum przyspieszyć męczące przepisywanie adresów.

Pozostaje tylko wyjaśnić, w jaki sposób korzystać z tego programu. Dla lepszego zilustrowania posłużę się przykładem.

Wyobraźmy sobie, że posiadamy monitor pamięci, który jako program znajduje się pomiędzy adresami $C000 a $D000. Z pewnych, jedynie nam wiadomych względów chcielibyśmy mieć ten sam monitor pod adresem $4000. Oto kolejność postępowania:

1. Wgrać przeznaczony do relokacji program do pamięci i obejrzeć go pod kątem wektorów i tablic adresów, których ani reassembler ani relocator nie rozpozna. Należy te miejsca w programie zapamiętać i odpowiednio poprawić w przyszłości;

2. Załadować do pamięci program „Relocator 64" i uruchomić go rozkazem RUN z poziomu interpretera BASIC;

3. Udzielić odpowiedzi na zadawane przez komputer pytania:

a) na „source start address" (adres początkowy programu do relokacji) odpowiadamy wpisując adress $C000;

b) na „sourse end address" (adres końca naszego programu) odpowiadamy wpisując $D000;

c) na „start of relocating area" (początek obszaru relokowanego).

Informacja ta jest potrzebna komputerowi do tego, aby relokować tylko te rozkazy procesora, które odnoszą się do wskazanego obszaru. W naszym przypadku obszarem tym jest miejsce zajmowane przez nasz monitor (zmiany muszą dotyczyć odwołań w nim samym).

Odpowiedź - $C000;

d) na „end of relocating area" (koniec obszaru relokowanego) odpowiadamy $CFFF;

e) na „destination address" (adres przeznaczenia) odpowiadamy $4000;

f) na „$01 value" (zawartość komórki $01-ustawienie bankowania pamięci podczas relokowania) wciskamy RETURN potwierdzając standardową wartośc $37;

g) na pytanie „A-automatic; M-manual" udzielamy odpowiedzi wciskając klawisz „a" (relokacja automatyczna) lub „m" (relokacja, w której komputer prosi o potwierdzenie relokacji każdego z relokowanych adresów)

Po udzieleniu prawidłowych odpowiedzi następuje operacja relokowania, podczas której na ekranie jest wyświetlana treść relokowanego programu w postaci mnemoników rozkazów procesora 6510. Pracę programu relokującego można przerwać klawiszem SPACE. Po zakończeniu pracy program wraca do poziomu BASIC'a a pod adresem przeznaczenia znajduje się przerelokowana wersja naszego monitora. Teraz należy jeszcze tylko uważnie poprawić wszystkie te miejsca, w których znajdują się dotyczące jeszcze starego obszaru tablice adresów lub ustawiania wektorów.

Paweł Sołtysiński

Flash

Posiadając naszego poczciwego Commodore 64 na pewno nie raz poczyniliśmy pewne spostrzeżenia względem innych komputerów, np. dlaczego nie ma w C-64 „pikającej" klawiatury, jak np. w ATARI, lub dlaczego nie ma funkcji Flash (mrugającego tekstu) jak np. w Spectrum czy IBM ?? O ile „udźwiękowiona" klawiatura doczekała się już kilku rozwiązań, to funkcji flash nie widziałem jeszcze w żadnym magazynie. Przedstawiony obok listing należy wpisać do pamięci komputera korzystając z programu KOREKTOR 64, a następnie nagrać gotowy program na nośnik za pomocą rozkazu:

S„flash",08,C000,C0C0 - dla dysku, lub:

S„flash",01,C000,C0C0 - dla taśmy.

Tak zapisany „Flash" wgrywać możemy do pamięci za pomocą rozkazu LOAD z ",1" po numerze urządzenia. Program należy uruchomić rozkazem SYS 49152 (lub SYS $C000) i od tej pory wszystkie znaki wyświetlone w negatywie będą ...mrugającymi pozytywami ! Jedynym kłopotem będzie trochę „nienormalny" wygląd kursora oraz fakt, że używając znaku spacji w mrugającym tekście należy go wprowadzić poprzez kombinację klawiszy Shift i Space.

```
:C000 78 A9 33 85 01 A9 D0 85
:C008 21 A9 D4 85 FC A2 04 A0
:C010 00 84 20 84 FB 84 FD 20
:C018 7B C0 A9 D8 85 21 A9 DC
:C020 85 FC A2 04 20 7B C0 A9
:C028 D0 85 21 A9 E0 85 FC A9
:C030 E4 85 FE A2 04 20 8C C0
:C038 A9 D8 85 21 A9 E8 85 FC
:C040 A9 EC 85 FE A2 04 20 8C
:C048 C0 A9 07 A2 07 9D 00 D5
:C050 9D 00 DD CA 10 F7 A9 37
:C058 85 01 AD 00 DD 29 FC 8D
:C060 00 DD A9 C4 8D 88 02 A9
:C068 15 8D 18 D0 A9 A1 8D 14
:C070 03 A9 C0 8D 15 03 20 44
:C078 E5 58 60 B1 20 91 20 91
:C080 FB C8 D0 F7 E6 21 E6 FC
:C088 CA D0 F0 60 B1 20 91 FB
:C090 A9 00 91 FD C8 D0 F5 E6
:C098 21 E6 FC E6 FE CA D0 EC
:C0A0 60 A9 00 F0 06 CE A2 C0
:C0A8 4C 31 EA A9 0E 8D A2 C0
:C0B0 AD 18 D0 49 0D 8D 18 D0
:C0B8 D0 EE 7C 00 00 00 00 00
```

Paweł Sołtysiński

KOREKTOR

Program ten ma na celu ułatwić wpisywanie i poprawianie zamieszczonych w naszym magazynie listingów programów w kodzie maszynowym. Z racji swej formy (długie kolumny cyfr) są one w oczywisty sposób męczące przy wpisywaniu, co sprzyja powstawaniu błędów.

Zamieszczony obok listing należy wpisać posługując się dowolnym monitorem pamięci począwszy od adresu $4000. Po uważnym wpisaniu należy go nagrać na posiadany nośnik z poziomu monitora, przy czym adres końca wynosi $4308. Tak nagrany program należy potem wgrywać do pamięci za pomocą dyrektywy LOAD w następujący sposób:

LOAD"korektor 64",1,1 - dla taśmy
lub
LOAD"korektor 64",8,1 - dla dysku.

Po wgraniu można wykonać komendę NEW, która ustawi nam wektory programu w języku BASIC na pozycję wyjściową, a potem uruchomić sam korektor za pomocą rozkazu SYS16384 (lub SYS $4000).

Obsługa programu przypomina trochę pracę na którymś z monitorów pamięci i „odwołuje się" do nawyków z nią związanych. A oto komendy KOREKTORA 64:

M - monitorowanie pamięci. Samo „m" powoduje wyświetlenie następnej strony pamięci; „m" z adresem (np. m4000) spowoduje rozpoczęcie monitorowania od wskazanego adresu. Każda z drukowanych na ekranie linii (po 8 bajtów) zawiera na końcu dwuznakowy kod kontrolny (w trybie INVERSE VIDEO), odpowiedni dla tych 8 bajtów.

I - wprowadzanie danych do pamięci. Samo „i" pozwoli kontynuować wprowadzanie od miejsca, w którym ostatnio je przerwaliśmy. Jeśli natomiast po literze „i" podamy adres (np. 10801) to rozpoczniemy wpisywanie danych od tego adresu. Wprowadzanie danych polega na przepisaniu 8 bajtów i wciśnięciu klawisza RETURN. Komputer wyświetli sumę kontrolną wprowadzonych liczb (w INVERSE VIDEO), którą należy porównać z tą wydrukowaną przy listingu w magazynie. Jeżeli porównanie wypadnie pomyślnie, należy zacząć przepisywać następną linię - w przeciwnym przypadku należy wstrzymać dalsze wprowadzanie danych przez wciśnięcie samego RETURN, po czym poprawić złą linię za pomocą komendy „i"+adres.

F - wypełnianie (fill) pamięci wskazaną wartością. Przed przystąpieniem do wpisywania programu wskazane jest wyczyszczenie (np. wypełnienie wartością zero) obszaru, w którym wprowadzony program będzie się znajdował. Składnia komendy FILL jest następująca:

F adres1,adres2,wartość

gdzie „adres1" to adres początkowy wypełnionego obszaru, „adres2" to adres końcowy obszaru a „wartość" to bajt wypełnienia.

Przykładowy zapis:

FC000,C500,00 - oznacza wypełnienie wartością zero obszaru od $C000 do $C4FF włącznie.

L - ładowanie do pamięci (load).
Składnia jak w większości monitorów:
L „nazwa",numer-urządzenia
S - nagrywanie na nośnik. Składnia:
S„nazwa",numer – urządzenia,adres1,adres2
gdzie „adres1" to początek nagrywanego obszaru a „adres2" to adres jego końca.
X - powrót do BASIC'a.

Przy pomocy KOREKTORA 64 można także w prosty sposób wykonywać wydruki własnych programów na drukarkę wraz z kodami kontrolnymi. W tym celu należy wykonać następującą sekwencję rozkazów BASIC:

OPEN1,4:CMD1

a potem uruchomić KOREKTOR i używając rozkazu „m" wydrukować żądany obszar przy pomocy drukarki.

Przedstawiany program zamieszczony został na marcowym dysku PDP.

```
:4000 AD 21 D0 29 0F 49 08 8D
:4008 86 02 A9 37 85 01 A9 18
:4010 A0 40 20 1E AB 4C EF 40
:4018 0D 0D 2A 2A 20 4B 4F 52
:4020 45 4B 54 4F 52 20 43 36
:4028 34 20 2A 2A 0D 41 55 54
:4030 4F 52 3A 50 2E 53 4F 4C
:4038 54 59 53 49 4E 53 4B 49
:4040 20 2F 20 43 36 34 2B 34
:4048 20 26 20 41 4D 49 47 41
:4050 0D 52 4F 5A 4B 41 5A 59
:4058 3A 20 49 2C 4D 2C 4C 2C
:4060 53 2C 46 2C 58 00 00 20
:4068 CF FF C9 0D D0 03 A9 80
:4070 60 C9 30 90 F9 C9 47 B0
:4078 F5 C9 3A 90 04 C9 41 90
:4080 ED 29 3F C9 30 B0 04 18
:4088 69 09 60 E9 30 60 20 67
:4090 40 10 01 60 0A 0A 0A 0A
:4098 8D 66 40 20 67 40 10 01
:40A0 60 0D 66 40 8D 66 40 2C
:40A8 8C 40 60 00 40 20 8E 40
:40B0 10 01 60 8D AC 40 20 8E
:40B8 40 30 F7 8D AB 40 AA AC
:40C0 AC 40 A9 00 60 48 4A 4A
:40C8 4A 4A 20 D0 40 68 29 0F
:40D0 C9 0A 90 05 18 69 37 D0
:40D8 02 09 30 4C D2 FF A9 3A
:40E0 D0 0A A9 20 D0 06 A9 0D
:40E8 D0 02 A9 3E 4C D2 FF 20
:40F0 E6 40 20 CF FF C9 0D F0
:40F8 F6 C9 58 D0 02 18 60 C9
:4100 4D D0 03 4C 2D 41 C9 49
:4108 D0 03 4C AF 41 C9 46 D0
:4110 03 4C 12 42 C9 53 D0 03
:4118 4C 9A 42 C9 4C D0 03 4C
:4120 E1 42 A9 3F 20 EC 40 4C
:4128 EF 40 06 28 41 20 AD 40
:4130 30 06 8E 2B 41 8C 2C 41
:4138 A9 15 8D 2A 41 AD 2B 41
:4140 85 FB AD 2C 41 85 FC 20
:4148 E6 40 20 DE 40 AD 2C 41
:4150 20 C5 40 AD 2B 41 20 C5
:4158 40 A0 00 20 E2 40 78 E6
:4160 01 B1 FB C6 01 58 20 C5
:4168 40 C8 C0 08 90 ED 20 E2
:4170 40 20 E2 40 A9 12 20 D2
```

```
:4178 FF A0 00 98 78 E6 01 18
:4180 71 FB C8 C0 08 90 F8 C6
:4188 01 58 20 C5 40 A9 92 20
:4190 D2 FF 98 18 6D 2B 41 8D
:4198 2B 41 90 03 EE 2C 41 20
:41A0 E1 FF F0 05 CE 2A 41 10
:41A8 94 4C EF 40 01 C8 00 20
:41B0 AD 40 30 06 8E AC 41 8C
:41B8 AD 41 20 E6 40 20 EA 40
:41C0 AD AD 41 85 FC 20 C5 40
:41C8 AD AC 41 85 FB 20 C5 40
:41D0 20 E2 40 A0 00 8C AE 41
:41D8 20 8E 40 30 3A 78 E6 01
:41E0 91 FB C6 01 58 18 6D AE
:41E8 41 8D AE 41 20 CF FF C8
:41F0 C0 08 90 E4 20 E2 40 A9
:41F8 12 20 D2 FF AD AE 41 20
:4200 C5 40 98 18 6D AC 41 8D
:4208 AC 41 90 03 EE AD 41 4C
:4210 BA 41 20 AD 40 10 03 4C
:4218 22 41 86 FB 84 FC 20 CF
:4220 FF 20 AD 40 30 F1 86 FD
:4228 84 FE 20 CF FF 20 8E 40
:4230 30 E5 85 02 78 E6 01 A0
:4238 00 A5 02 91 FB E6 FB D0
:4240 02 E6 FC A5 FC C5 FE D0
:4248 F0 A5 FB C5 FD D0 EA C6
:4250 01 58 4C EF 40 68 68 4C
:4258 22 41 4B 4F 52 45 4B 54
:4260 4F 52 20 36 34 20 20 20
:4268 20 20 0B 08 20 CF FF C9
:4270 22 D0 E2 A0 00 20 CF FF
:4278 C9 22 F0 0F 99 5A 42 C8
:4280 C0 10 90 F1 20 CF FF C9
:4288 22 D0 CA 8C 6A 42 20 CF
:4290 FF 20 8E 40 30 BF 8D 6B
:4298 42 60 20 6C 42 20 CF FF
:42A0 20 AD 40 30 B2 86 FB 84
:42A8 FC 20 CF FF 20 AD 40 30
:42B0 A6 8E D4 42 8C D6 42 20
:42B8 E6 40 A9 80 85 9D AE 6B
:42C0 42 A9 08 A8 20 BA FF AD
:42C8 6A 42 A2 5A A0 42 20 BD
:42D0 FF A9 FB A2 08 A0 43 C6
:42D8 01 20 D8 FF E6 01 4C EF
:42E0 40 20 6C 42 20 E6 40 AE
:42E8 6B 42 A9 08 A8 20 BA FF
:42F0 A9 80 85 9D AD 6A 42 A2
:42F8 5A A0 42 20 BD FF A9 00
:4300 20 D5 FF 4C EF 40 41 00
:4308 9F 00 00 00 00 00 00 00
```

Paweł Sołtysiński

SCULPT ANIMATE 4D (cz.2)

W pierwszym odcinku opisu Sculpt'a (miał on być próbą zachęcenia czytelników do pracy z tym narzędziem) omówiłem jego cechy, wady i zalety, dzisiaj zajmę się pracą z tym programem.

Po uruchomieniu Sculpt'a mamy do dyspozycji trzy okna oraz pięć menu. Te trzy okna to widok naszej przestrzeni w trzech rzutach z dołu, północy oraz z zachodu. Kierunek rzutu w jednej płaszczyźnie tj. Dół-Góra, Północ-Południe, Zachód-Wschód, możemy zmienić za pomocą gadżetu nr 3. Litery znajdujące się przy krawędziach okna oznaczają kierunek, który znajduje się z danej strony okna np. jeżeli mamy rzut z dołu (DOWN) to u góry będziemy mieć Północ (N). Jeżeli chcemy poruszać się po dosyć dużej przestrzeni jaką sobie zdefiniowaliśmy to musimy użyć strzałek kierunkowych znajdujących się obok liter oznaczających kierunek. Gadżety nr 8 i 9 służą do pomiejszania i powiększania obszaru widocznego w oknie. Gadżet nr 5 służy do centrowania okna tj. przesuwa przestrzeń tak, aby kursor znalazł się w jego środku.

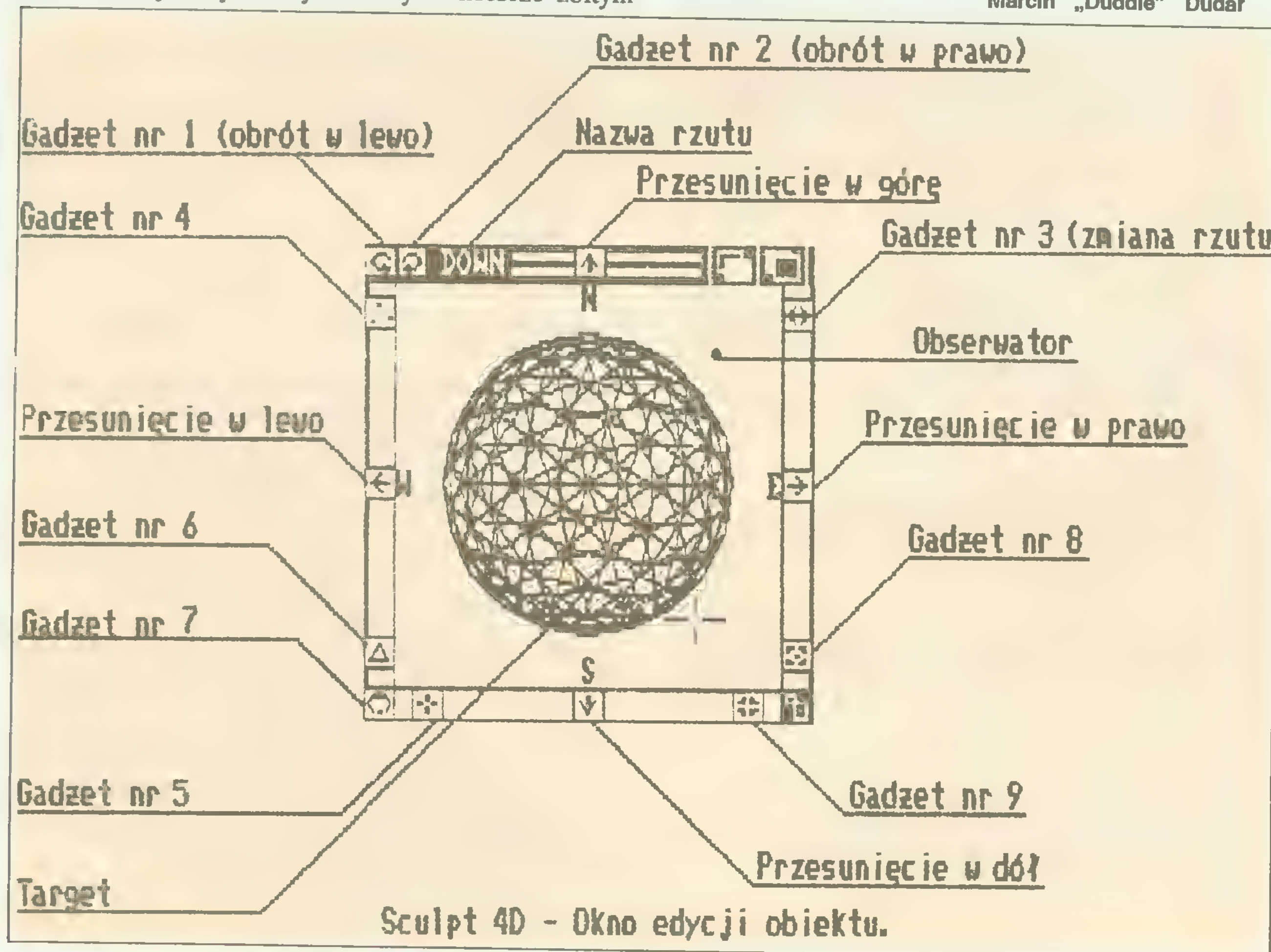
Każdy obiekt w programie Sculpt 4D składa się z płaszczyzn, które są wyznaczone przez trzy połączone punkty, a więc są trójkątami (np. kwadrat budowany jest z dwóch trójkątów). Punkty (wierzchołki) możamy stawiać wciskając lewy a następnie prawy przycisk myszy. Punkt zostanie postawiony tam gdzie aktualnie znajduje się kursor (krzyżyk - nie mylić z pointer'em myszy !). Gdy na postawionym punkcie klikamy dwa razy lewym przyciskiem to zmienia on kolor na różowy lub żółty. Kolor różowy oznacza, że punkt jest nie aktywny czyli nie może być edytowany. Punkty w kolorze żółtym możemy przesuwać, łączyć, wymazywać, etc. Jeżeli chcemy uaktywnić wszystkie widoczne punkty to klikamy myszą na gadżecie nr 4. Jeżeli wszystkie punkty miały kolor różowy (nieaktywne) to zmieniają kolor na żółty (aktywne) i vice versa. Aby uaktywnić punkt pod kursorem możemy dokonać tego wciskając klawisze AMIGA + Y, ażeby zdeaktywować - AMIGA + P. Dwa aktywne punkty możemy połączyć kreską za pomocą gadżetu nr 6. Najpierw deaktywujemy wszystkie punkty (gadżet 4), a potem uaktywniamy dwa punkty, które chcemy połączyć przy użyciu gadżetu nr 6.

Obracania przestrzeni dokonujemy za pomocą gadżetów 1 i 2. Wszystkie punkty aktywne (żółte) zostaną obrócone względem kursora w zadanym kierunku. Za pomocą gadżetu nr 7 możemy dokonać przesunięcia aktywnych punktów. Po uaktywnieniu punktów i wciśnięciu tego gadżetu przesuwamy kursor o tyle o ile chcemy przesunąć punkty i wciskamy lewy przycisk myszy w nowym położeniu. Gdy nowa lokacja nam odpowiada wciskamy prawy przycisk myszy, co zakończy możliwość przesuwania punktów.

Na rysunku są także widoczne punkty Obserwator i Target (cel). Punkt Obserwator jest to punkt, z którego będziemy obserwować naszą przestrzeń, a punkt Target jest to punkt, na który będziemy patrzeć. Wyznaczają one kierunek patrzenia i perspektywę. Aby ustawić punkt obserwatora musimy ustawić na wybranej pozycji a następnie wybrać z menu OBSERWER opcję Location lub wcisnąć AMIGA + O, natomiast aby ustawić punkt docelowy (Target) ustawiamy kursor w wybranym miejscu w przestrzeni i wybieramy z menu OBSERWER opcję Target.

Za miesiąc będziecie mogli przeczytać o opcjach menu.

Marcin „Duddle" Dudar



Sculpt 4D - Okno edycji obiektu.

Kącik początkującego kodera (cz. 2)

W ostatnim odcinku przedstawiłem na przykładzie rozkazu „MOVE" tryby adresowania mikroprocesora MOTOROLA 68000. Dzisiaj poznamy kolejne rozkazy oraz zapoznamy się z działaniem krótkich procedur napisanych w języku assemblera.

Opisując rejestry mikroprocesora wspomniałem o tzw. rejestrze statusu zawierającym zestaw znaczników. Modyfikowane są one po wykonaniu większości komend, z których składa się nasz program, w zależności od ostatecznego wyniku. Rozpatrzmy teraz następującą sekwencję rozkazów. Po prawej stronie przedstawię w jaki sposób zmieniać się będą przykładowe dwa znaczniki po wykonaniu danej instrukcji:

```
move.b  #$10,d0  ;Z=0 Liczba różna od zera, N=0 nieujemna
sub.b   #$10,d0  ;Z=1 Liczba równa zero,    N=0 nieujemna
sub.b   #$01,d0  ;Z=0 Liczba różna od zera, N=1 ujemna
```

W ten właśnie sposób znaczniki charakteryzują wynik każdej operacji. Ich znajomość jest konieczna programiście, dlatego też przy omawianiu rozkazów będę pokrótce charakteryzował zachowanie znaczników. W naszym przykładzie ostatecznie otrzymaliśmy liczbę $ff tj. -1, gdyż mikroprocesor liczby z ustawionym najstarszym bitem (w tym wypadku 7, ponieważ operujemy na bajcie) interpretuje jako ujemne. Na bajcie możemy więc przedstawić liczbę z zakresu od -128 do 127.

Rozkazy mikroprocesora MOTOROLA 68000 zasadniczo dzielą się na 8 grup:

1. Rozkazy przesyłania danych.
EXG, LEA, LINK, MOVE, MOVEM, MOVEP, MOVEQ, PEA, SWAP, UUNLK.
Rozkazy te służą do przemieszczania danych w pamięci komputera.

2. Rozkazy arytmetyki.
ADD, ADDA, ADDI, ADDQ, ADDX, CLR, CMP, CMPA, CMPI, CMPM, DIVS, DIVU, EXT, MULS, MULU, NEG, NEGX, SUB, SUBA, SUBI, SUBQ, SUBX, TAS, TST.
Rozkazy te służą do wykonywania podstawowych działań arytmetycznych takich jak dodawanie, odejmowanie, mnożenie, i dzielenie ze znakiem lub bez. Poza tym rozkazów tych używamy przy porównaniach, testowaniach i rozszerzeniach liczb.

3. Rozkazy logiki.
AND, ANDI, OR, ORI, EOR, EORI, NOT.
Są to rozkazy logicznych operacji na liczbach.

4. Rozkazy przesunięć i obrotów.
ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR.

5. Rozkazy manipulacji bitami.
BTST, BSET, BCLR, BCHG.
Rozkazy te wykonują operacje na poszczególnych bitach.

6. Instrukcje BCD.
ABCD, SBCD, NBCD.
Są to instrukcje działań na tzw. kodzie BCD.

7. Rozkazy sterowania programem.
Bxx, DBxx, Sxx, BSR, JSR, RTS, JMP, RTR.
W skład tej grupy wchodzą rozkazy skoków, skoków warunkowych, skoków do podprogramów i powrotu z nich oraz rozkazy obsługi przerwań.

8. Rozkazy kontroli systemu.
MOVE USP, RESET, RTE, STOP, CHK, TRAPV, TRAP.
Rozkazy te służą do kontroli mikroprocesora.

Przedstawię teraz szczegółowy opis kilku wybranych instrukcji. Zostały one tak dobrane, abyśmy mogli w oparciu o nie próbować tworzyć własne procedury.

CLR <adres efektywny>(Clear an Operand- czyszczenie operandu).

Instrukcja ta powoduje wyzerowanie operandu określonego odpowiednim trybem adresowania. Może przyjmować rozmiar b,w lub l czyli bajtu, słowa lub długiego słowa.

Znaczniki: X - bez zmian, N=0, Z=1, V=0, C=0

Przykład:

```
clr.w  d0        - zostanie wyzerowane
                   młodsze słowo rejestru d0
clr. b $50000    - wyzeruje komórkę $50000.
```

LEA <adres efektywny>, Ax (Load Effective Address - załaduj adres efektywny).

Instrukcja ta umieści w rejestrze adresowym adres efektywny. Zostanie zmienione długie słowo.

Znaczniki: wszystkie pozostaną niezmienione.

Przykład:

```
lea $60124,a0  - w a0 zostanie umieszczony adres
                 $60124
```

CMP <adres efektywny>, Dx (Compare - porównaj).

Instrukcja ta porówna zawartość rejestru danych z operandem określonym przez adres efektywny. Odejmie operand źródłowy od rejestru danych. Wyniku tej operacji nie zapisze , wpłynie natomiast na znaczniki. Rozmiarem może być bajt, słowo, lub długie słowo. Instrukcja ta w połączeniu z rozkazami skoków warunkowych umożliwia programiście sterowanie programem po sprawdzeniu dowolnych warunków.

Znaczniki: X - bez zmian
N - ustawiany, gdy wynik ujemy

Z - ustawiany, gdy wynik równy jest zero - operand źródłowy jest równy operandowi przeznaczenia
V - ustawiany, gdy wystąpił nadmiar
C - ustawiany, gdy wystąpiła pożyczka

W przeciwnych wypadkach od wyżej opisanych, znaczniki zostaną wyzerowane.

Przykład:

cmp. b d1, d0 - zawartość d0 zostaje porównana z zawartością d1.
cmp. w $20000, d1 - słowo spod adresu $20000, zostanie porównane z zawartością d1.

CMPA <adres efektywny>, Ax (Compare address - porównaj adres).

Analogicznie jak CMP, z tą różnicą, iż operandem przeznaczenia jest rejestr adresowy, a nie danych.

Znaczniki: Tak samo jak CMP

Przykład:

cmp. w d0, a6 - porówna zawartość obu rejestrów.

CMP I #< liczba>,<adres efektywny> (Compare Immediate - porównanie natychmiastowe).

Analogicznie jak CMP, jednak porównana zostanie natychmiastowa liczba z operandem określonym adresem efektywnym.

Znaczniki: Tak samo jak CMP

Przykład:

cmpi. 1 #$12345678, (a0) - $12345678 porównana zostanie z liczbą znajdującą się pod adresem wskazanym przez rejestr a0.

CMPM (Ax)+, (Ay)+ (Compare Memory - porównanie pamięci

Instrukcja ta porównuje dwie liczby znajdujące się w pamięci pod adresami wskazanymi przez rejestry adresowe. Odpowiednio modyfikowane są znaczniki, następnie rejestry zwiększane są o 1,2 lub 4 w zależności od tego czy porównujemy bajt, słowo czy długie słowo. Instrukcji tej używamy przy porównaniu większych obszarów pamięci.

Znaczniki: Tak samo jak CMP

Przykład:

cmpm. 1 (a5)+, (a6)+

Bxx <etykieta> (Branch Conditionally - skok warunkowy)

Instrukcja Bxx jest instrukcją skoku warunkowego. Testuje wpierw określone znaczniki, a następnie, jeżeli zadany warunek jest spełniony wykonuje skok do etykiety. Istnieje 15 rodzajów instrukcji Bxx.

BCC - skok, gdy C=0.
BCS - skok, gdy C=1.
BEQ - skok, gdy Z=1. Wykona skok, jeżeli wynikiem ostatniej operacji było zero.
BGE - skok, gdy znaczniki N i V są ustawione lub wyzerowane.
BGT - skok, gdy N i V są ustawione, a Z wyzerowane albo N,V,Z są wyzerowane.
BHI - skok, gdy C i Z są wyzerowane.
BLE - skok, gdy Z=1 albo [N=1 i V=0] albo [N=0 i V=1].
BLS - skok, gdy C=1 albo Z=1.
BLT - skok, gdy [N=1 i V=0] albo [N=0 i V=1]
BMI - skok, gdy N=1. Wykona skok, jeżeli wynikiem ostatniej operacji była liczba ujemna.
BNE - skok, gdy Z=0. Wykona skok, jeżeli wynikiem ostatniej operacji była liczba różna od zera.
BPL - skok, gdy N=0. Wykona skok, jeżeli wynikiem ostatniej operacji była liczba dodatnia.
BVC - skok, gdy V=0
BVS - skok, gdy V=1
BRA - skok ten zostanie wykonany zawsze, gdyż nie testowane są żadne znaczniki.

Znaczniki: Żadne znaczniki nie są modyfikowane.

Przykład:

beq tam - jeżeli znacznik Z był ustawiony, to licznik programu zostanie ustawiony na adres wskazany przez etykietę „tam", w przeciwnym wypadku mikroprocesor przejdzie do wykonania instrukcji znajdującej się bezpośrednio za „bne tam".

Teraz wykorzystując poznane rozkazy zapoznamy się z prostymi, przykładowymi procedurami. Pierwsza z nich służy do przepisywania obszarów pamięci. Poniższa procedura skopiuje blok pamięci o adresach $50000-$50100, pod adresy $60000-$60100. Odpowiednio zmieniając dane dla programu, możemy kopiować obszary o dowolnej długości i zaczynających się od dowolnych adresów.

```
      lea $50000,a0  ; adres miejsca skąd przepisujemy
      lea $60000,a1  ; adres miejsca gdzie przepisujemy
loop: move.b   (a0)+,(a1)+  ; bajt spod adresu wskaza
                      nego przez rejestr a0 przepisujemy
                      pod adres wskazywany przez rejestr
                      a1,następnie oba rejestry zostaną
                      zwiększone o 1.
      cmpa.1 #$50101,a0 ;czy w a0 znajduje się adres
                      $50101?
      bne loop       ; jeżeli nie to przejdą do etykiety
"loop"
      rts            ; powrót
```

Następna procedura służy do sumowania kolejnych liczb naturalnych. Sumuje liczby od 0 do 10 i uzyskany wynik umieszcza w rejestrze d1.

```
      clr.1    d0     ; czyścimy d0, który wykorzystamy
                      jako licznik pętli
      clr.1    d1     ; czyścimy d1, w którym
                      przechowywać będziemy sumę.
tutaj: add. 1  d0,d1  ; do zawartości d1 dodajemy zawa-
                      rtość d0, wynik otrzymujemy w d1.
      addq.1   #1,d0  ; zwiększamy o 1 d0: następna licz
                      ba do dodania
      cmpi.b   #11,d0 ; czy w d0 jest 11? W tym miejscu
                      podajemy ostatnią liczbę, którą
                      chcemy zsumować powiększoną o 1.
                      W naszym przypadku sumujemy do
                      10 więc podajemy 11.
      bne      tutaj  ; jeżeli zawartość d0 różna od 11 to
                      skok do etykiety „tutaj".
      rts             ; powrót, a wynik w d1
```

Jeżeli chcielibyśmy sumować nie od 0, a od np. 5 to zamiast instrukcji „clr.1 d0" należałoby użyć „move.1 #5,d0".

Krzysztof „K.K." Kobus

Amos Creator V1 - opis systemu (cz. 1)

Basic to język pełny ograniczeń, nieprzydatny do profesjonalnych zastosowań. Stykając się z interpreterami na maszyny 8-bitowe czy choćby osławionym Microsoft Basic z dysku Extras można raz na zawsze zniechęcić się do Basic'a. Szczególnie ten ostatni - pełny błędów, wolny, nieprzyjazny dla użytkownika - edytorem może skutecznie odstraszyć od programowania.

Złą passę Basic'a na Amigę przełamał swego czasu słynny GFA-Basic oferując bogactwo funkcji, dobry edytor, kompilator. Wydawało się że jest to wszystko co można zrobić z Basic'em. I oto... cóż za zaskoczenie ! Pojawia się nowy interpreter Basic'a o rewelacyjnych możliwościach przebijających wszelką konkurencję w tej dziedzinie. Nazwa nowego potentata, jedynego w swojej klasie brzmi:

AMOS !

Mało jest na Amidze tak starannie opracowanego oprogramowania użytkowego. Atrakcyjna szata graficzna, niezły edytor, wspaniałe wbudowane możliwości graficzne i muzyczne, wspomaganie Intuition, bespośrednie sterowanie hardwar'em czynią ten program poważnym konkurentem innych języków programowania. Według autorów AMOS'a służyć ma do tworzenia gier, demonstracji, oprogramowania edukacyjnego. AMOS wykorzystuje bardzo aktywnie wszelkie sprzętowe możliwości Amigi i dzięki temu jest programem unikalnym.

Twórcą AMOS'a jest Francois Lionet (Francja) współpracujący z firmą Mandarin Software. Jego poprzednim bardziej znanym programem jest STOS - bardzo szybki Basic napisany dla Atari ST. AMOS jest zbliżony koncepcyjnie do STOS'u lecz nie jest jedynie mechanicznie przekodowany. Niemalże wszystkie funkcje zostały napisane od nowa z wykorzystaniem specyficznych cech Amigi. Natomiast nowy program tegoż autora tj. STOSPro jest próbą implementacji AMOS'a na ST. Jednak jak wszyscy wiedzą ST nie ma właściwie specjalizowanych układów więc nie należy się spodziewać że STOSPro przerośnie swego poprzednika.

AMOS to nowa koncepcja Basic'a (czy jeszcze Basic'a?). A to niektóre z licznych cech tego wspaniałego programu:

- strukturalność,
- procedury/funkcje,
- tzw. auto ident,
- szybki edytor,
- tryb bezpośredni,
- szybki system okien,
- aktywne użytkowanie blittera i copper'a (muzyka, animacja),
- łatwe do oprogramowania sprite'y, bob'y, dual playfield,
- profesjonalny system menu,
- kilka programów pamięci,
- akcesoria wspomagające napisane w AMOS'ie,
- skromne wymagania sprzętowe (0.5MB, 1 dysk),
- własne biblioteki - rozszerzenia.

Częstym zarzutem stawianym Basic'owi jest brak strukturalnej budowy programu. AMOS całkowicie opiera się temu pomówieniu. Programy można pisać zarówno z, jak i bez GOTO. Programista w C, Pascal'u czy Fortranie nie powinien mieć kłopotów z adaptacją do konwencji składniowych AMOS'a. Procedury, funkcje wyliczane, przysłanianie zmiennych czy pętle warunkowe to mechanizmy nieobce AMOS'owi. Dla laika: AMOS różni się od Pascala prostszą budową - brak w nim struktur blokowych i rozbudowanych typów danych. Przebija jednak Pascal swoją prostotą, możliwością intrpretacji i bogactwem funkcji niskiego poziomu.

Budowa systemu

Pierwszym co ukazuje się po załadowaniu AMOS'a do pamięci jest ekran systemowy. Dzieli się on na dwie części: górną, stanowiącą obszar menu oraz dolną - obszar edytora pełnoekranowego.

Menu

Górny obszar przeznaczony jest do wydawania komend AMOS'owi. Znajduje się tam w dwóch liniach 10 pól z krótkimi komendami (Run, Test, Indent...). Wywołuje się je w dwojaki sposób. Po pierwsze, można kliknąć na danym polu lewym klawiszem myszy. Drugą metodą jest wciśnięcie odpowiedniego klawisza funkcyjnego. Górnej linii (tj. Run, Test...) odpowiadają klawisze F1..F5 zaś dolnej F6..F10. Poza menu widocznym na ekranie istnieje jeszcze pięć przełączanych zestawów. Można je uzyskać przytrzymując dodatkowo klawisz Ctrl, Shift, Alt, lewą lub prawą Amigę. Klawisz Shift (dla niezorientowanych - to ten pod klawiszem Ctrl lub Return) może być zastąpiony prawym klawiszem myszy. Najważniejsze dla początkujących klawisze to Run = F1 (uruchamia program), Load = Shift F1 (załadowanie programu), Save = Shift F2 (nagranie programu na dysk) ewentualnie Insert Line = F10 (wstawia nową linię). Menu Ctrl to tak zwane 'Block menu' zaś Alt 'Search menu'. Pierwsze jest odpowiedzialne za operacje na blokach tekstu, drugie za poszukiwanie w tekście wzorców ew. wymianę na inne. Pod klawiszami Amiga (po lewej i prawej stronie klawisza spacji) znajdują się komendy AMOS'a. Zamiast wpisywać całą komendę wystarczy wcisnąć odpowiedni klawisz funkcyjny razem z Amigą.

Edytor

Na szczycie okna edytora znajduje się linia statusowa. Znajdują się tam takie informacje jak położenie kursora (linia i kolumna), aktualne tryby wprowadzania znaków (Insert/Overwrite, Caps Lock), ilość wolnej pamięci, rozmiar tekstu programu i jego nazwa. Obsługa edytora jest zbliżona do obsługi Cygnus'a czy innego edytora pełnoekranowego. Próbując kombinacje klawiszy kursora z Ctrl, Shift i Alt szybko poznasz ich znaczenie. Jedynym odstępstwem jest kasowanie linii (Ctrl Z) popularne w edytorach IBM PC. Ważną cechą edytora jest tzw. Auto Identify. Polega to na tym, ze po wprowadzeniu linii automatycznie rozpoznawane są nazwy

komend, zmiennych i procedur. Linia zostaje poddana automatycznie ponownej edycji, polegającej na wcięciu zbędnych spacji, zmianie liter: w zmiennych i nazwach procedur na duże, zaś nazwach komend na małe (poza pierwszą literą).

Edytor posiada jedną czasami kłopotliwą cechę - po wciśnięciu klawisza Caps Lock nie działają klawisze kursora i backspace.

Bloki tekstu można zaznaczyć przy pomocy prawego klawisza myszy. Jest to metoda alternatywna do Block menu.

Klawisz Esc i Help mają specjalne znaczenie. Ten pierwszy pozwala na przechodzenie do i powrót z trybu bezpośredniego wprowadzania komend. Tryb ten umożliwia 'ręczne' sprawdzanie działania komend głównych programów systemu.

Tryb wykonywania programu

Trwa od uruchomienia opcji Run do zakończenia programu bądź to przez dotarcie do końca tekstu źródłowego, bądź po komendzie Stop, bądź po wciśnięciu Ctrl C (przerwanie wykonania programu). W tym czasie widoczne są wszystkie efekty wywołane przez program.

Tryb bezpośredni

Wchodzimy do niego przez wciśnięcie Esc, ewentualnie po zakończeniu wykonywania programu. Wciskając klawisze kursora góra i dół przemieszczamy okno do wprowadzania komend. Odpowiednio po przytrzymaniu Shift'a możliwe jest powiększanie/pomniejszanie tegoż okna. Wciśnięcie Help wywołuje na ekran listę komend dostępnych z klawiatury po wciśnięciu Amigi i odpowiedniego klawisza funkcyjnego. Wciskając F1 wywołujemy ostatnio użytą komendę, F2 służy do wywołania przedostatniej itd.

Język

Dlaczego AMOS nazywamy w ogóle Basic'em? Przypuszczalnie dlatego, że jest inerpreterem i posiada instrukcję GOTO. Linie mogą być numerowane, choć nie muszą. Etykiety (zamiast numerów linii) wprowadza się przez napisanie ich jako pierwszych w linii i postawieniu po nich dwukropka. W linii można umieścić wiele komend dzieląc je dwukropkami. Właściwie wszystkie komendy standardowego Basic'a działają tak samo więc skupimy się jedynie na nowościach.

Ekran: standardowo otwierany jest w niskiej rozdzielczosci 320*200 punktów w szesnastu kolorach. Zmienić to można wprowadzając komendę otwierającą inny ekran:

Screen Open N, W, H, C, M
N - numer ekranu (np. 1)
W - szerokość ekranu (np. 640)
H - wysokość ekranu (np. 256)
C - ilość dostępnych kolorów (np. 4)
M - rozdzielczość (np. Hires (Lowres)).

Wystarczy pozmieniać trochę parametry, aby zorientować się w działaniu tej komendy. Wszelkie doświadczenie w tej dziedzinie najlepiej czerpać z praktyki.

AMOS dostarcza kilka przydatnych w programowaniu komend. Często trzeba przerwać/kontynuować działanie programu po wciśnięciu dowolnego klawisza lub myszy. Przydatne tu będą funkcje:

Wait Key - czeka na klawisz

Mouse Key - zwraca 0, gdy nie wciśnięty klawisz myszy np.

Until Mouse Key=0 : Wend - oczekuje wciśnięcia myszy.

Ten ostatni przykład demonstruje wspaniałą cechę AMOS'a - strukturalność. Na początek opiszemy znaną komendę IF - rozgałęzienie warunkowe programu. Syntaktyka jest następująca:

```
If warunek Then
k11
Komendy wykonywane, jeśli warunek jest prawdziwy
k12
...
k1n
Else
k21
Komendy wykonywane, jeśli warunek jest fałszywy
k22
...
k2n
End If
```

Przykład:

```
If Mouse Key=1 Then
  Print „Lewy klawisz myszy"
Else
  Print „Nie przycisnąłeś lewego klawisza myszy"
End If
```

Kolejną ważną komendą jest instrukcja cyklu FOR. Jej syntaktyka jest identyczna jak w innych implementacjach:

```
For licznik = początek To koniec Step krok
  k1
  ..
  kn
Next
```

Zwróć uwagę, że Next nie potrzebuje nazwy zmiennej licznikowej. Składnik Step jest opcjonalny.

Niezwykle użyteczna jest komenda WHILE dobrze znana z C, Pascala i innych języków kompilowanych wysokiego poziomu:

While warunek - dopóki warunek jest spełniony wykonuj komendy zawarte od While do Wend

```
  k1
  ..
  kn
Wend
```

Podobna jest pętla Repeat:

Repeat - powtarza operacje zawarte między Repeat a Until aż warunek zostanie spełniony

```
k1
...
kn
Until warunek
```

Przykład:

```
Repeat
Input „Wprowadź T lub N", A$
Until A$=„T" or A$=„N"
```

Za miesiąc weźmiemy się za procedury, zmienne i być może grafikę. Życzę powodzenia w zmaganiach z AMOS'em !

Krzysztof Morol

GRACZ DOSKONAŁY

Witam po raz kolejny w naszej rubryce dla graczy-leniwców, gdzie znów nadgryziemy parę gier bez używania ostrych narzędzi.

AFTERBURNER

W czasie gry wpisz „TOGETHER IN ELECTRIC DREAMS". Teraz używając klawiszy możesz:
< - przejść do poprzedniej strefy,
> - przejść do następnej strefy,
G - uzyskać więcej rakiet,
T - uzyskać mniej rakiet,
N - dodatkowe życie,

By unikać rakiet leć u góry ekranu. W strefie 8 i 17 zwolnij by ominąć skały. By unikać rakiet sterowanych podczerwienią leć z pełną prędkością.

A.P.B.

Jeżeli w czasie jazdy będziesz trzymał wciśnięty przycisk, to wtedy włączysz syrenę przez co inne samochody będą zjeżdżały ci z drogi. Teraz bez obaw możesz rozwinąć pełną prędkość.

BEACH VOLLEYBALL

Wpisz „DADDY BRACEY" by włączyć opcję przełączania poziomów. Jeżeli obraz mrugnie, to używając klawisza F1 możesz zmieniać poziomy.

CAPONE

By znaleźć specjalny bonus poczekaj, aż znajdziesz się przed pocztą. Teraz zestrzel kulę na szczycie masztu, a liczba twoich punktów wzrośnie. Jeżeli strzelisz do niej jeszcze raz zobaczysz ekran przygotowany przez programistów. Jeżeli jeszcze raz do niej strzelisz wtedy staniesz się nieśmiertelny i zostaniesz przeniesiony do banku. Po przejściu tej strefy rozpoczniesz ją od początku, ale tym razem z warp speed.

CARRIR COMMAND

Zatrzymaj grę i wpisz „THE BEST IS YET BE". Teraz wciskając klawisz plus na klawiaturze numerycznej spowodujesz uodpornienie samolotów (manta) i poduszkowców (valrus) na wszystko z wyjątkiem zderzeń (low altitude).

Możesz też spróbować się poddać i za pomocą klawiszy „+" i „-" oglądnąć wszystkie obiekty występujące w grze.

CLOWN'O MANIA

W czasie gry wciśnij klawisz „HELP" by uzyskać więcej strzałów i skoków.

CYBERNOID

Na tytułowym obrazku wpisz „RAISTLIN" i wciśnij klawisz spacji by uzyskać nieskończoną ilość statków. Wciskając „N" przenosisz się do kolejnej strefy.

Możesz też spróbować zdefiniować klawisze jako „YX E S" by uzyskać nieskończoną ilość statków.

CYBERNOID II

Na tytułowym obrazku wpisz „NECRONOMICON" by uzyskać nieskończone życie. Zatrzymaj grę i wciśnij „N" by przejść do następnego poziomu lub wciśnij „L" by zacząć obecny poziom od początku.

CHASE HQ

W czasie gry przyciśnij lewy przycisk myszy, przycisk w joystick'u i wpisz „GROWLER". Teraz wciskając „T" odnawiasz zapas czasu. Na początku każdego poziomu wciśnij szybko spację by uzyskać dodatkowe nitro.

Jeżeli masz kłopoty ze sterowaniem, spróbuj użyć klawiszy „Z" i „X".

Mr. Raf.

Zmagania z Copperem (cz.4)

W poprzednim odcinku omówiliśmy instrukcje Copper'a. Korzystając z nich możemy pisać programy dla Copper'a.

Korzystając z instrukcji SKIP możemy uzyskiwać efekty pętli lub wywołania innych części programu. Wykonywanie różnych części CopperList'a odbywa się np. na zasadzie zmiany jego adresu początkowego (COP1LC). Jednak CL umieszczony od nowego adresu będzie wykonywany dopiero po zakończeniu kreślenia „ramki", czyli po wywołaniu przerwania wygaszania pionowego. Jeżeli jednak chcemy jakiś kawałek CL wykonać podczas wykonywania innego, wtedy możemy użyć instrukcji SKIP. Instrukcja SKIP jest podobna w działaniu do instrukcji WAIT z tą jednak różnicą, że Copper nie zatrzmuje się na tej instrukcji, ale wykonuje CL dalej. Jedynie jeżeli pozycje pionowa i pozioma są większe od pozycji podanych w instrukcji to następna instrukcja zostanie pominięta. W ten sposób można na przykład zbudować pętlę.

```
     move.1      #Copper, $dff080  ; wpisanie początku
                                     CL do COP1LC
     rts
Copper
     dc. w  $009c,$8010   ; ustawienie bitu przerwania CL
     dc. w  $0007,$0006   ; czekanie na pozycję poziomą
     dc. w  $0180,$0f00   ; zmiana koloru tła na czerwony
     dc. w  $0055,$0054   ; czekanie na pozycję poziomą
     dc. w  $0180,$00f0   ; zmiana koloru na zielony
     dc. w  $7f01,$ff01   ; pomiń następną instrukcję
                            jeżeli  VP=$7f
     dc. w  $0088,$0000   ; COPJMP1 - skocz do początku
                            CL
     dc. w  $ffff,$fffe   ; koniec
```

Właściwie wykorzystana instrukcja SKIP może przynieść wiele pożytku. Należy pamiętać, że taki program jest bardzo czasochłonny. Jeżeli zależy nam na szybkości wtedu nie powinniśmy zważać na pamięć ani na zasady dobrego (przejrzystego) programowania, ale na to, aby program był czasowo optymalny. Wszelkiego rodzaju pętle, odwołania i skoki zajmują wiele czasu i odnosi się to nie tylko do Copper'a, ale także do „normalnego" procesora. Gdy chcemy uzyskać super-szybką procedurę, to nalezy napisać ją „ciurkiem" tj. instrukcja za instrukcją, z pominięciem wszelkiego rodzaju pętli. W ten sposób powinien być napisany każdy CL - z minimalną ilością instrukcji WAIT oraz bez instrukcji SKIP.

Jeśli chcemy uzyskać bardzo popularny ostatnio w demach efekt plazmy, to musimy zbudować specjalny CopperList. Na każdą linię oczekujemy instrukcją Wait a potem wpisujemy kolejno np. 20 instrukcji zmiany koloru tła. Jeden taki krótki pasek koloru będzie miał długość około 8 pixeli. W ten sposób możemy cały ekran pokryć siatką takich pasków i tylko od szybkości oraz możliwości procedury będzie zależał wygląd takiej plazmy. Niektórzy programiści, aby przyspieszyć swój program, do zmiany kolorów używają blittera. Taką plazmę można oprzeć nie tylko na zasadzie zmiany kolorów, ale także na ich mieszaniu. Przykładowa procedura plazmy będzie zamieszczona na piątym dysku Public Dommain.

Na zakończenie, krótka procedura ustawiania „barów" czyli kolorowych pasków na ekranie.

```
move. 1   #Copper,$dff080    ; wpisanie początku CL
                               do  COP1LC
   rts
Copper
     dc. w  $0100,$0000      ; włączenie ekranów
     dc. w  $0180,$0000      ; kolor tła - czarny
     dc. w  $8007,$fffe      ; WAIT
     dc. w  $0180,$0333      ; zmiana koloru tła
     dc. w  $8107,$fffe      ; WAIT
     dc. w  $0180,$0555      ; zmiana koloru tła
     dc. w  $8207,$fffe      ; WAIT
     dc. w  $0180,$0888      ; zmiana koloru tła
     dc. w  $8307,$fffe      ; WAIT
     dc. w  $0180,$0ccc      ; zmiana koloru tła
     dc. w  $8407,$fffe      ; WAIT
     dc. w  $0180,$0fff      ; zmiana koloru tła
     dc. w  $8507,$fffe      ; WAIT
     dc. w  $0180,$0ccc      ; zmiana koloru tła
     dc. w  $8607,$fffe      ; WAIT
     dc. w  $0180,$0888      ; zmiana koloru tła
     dc. w  $8707,$fffe      ; WAIT
     dc. w  $0180,$0555      ; zmiana koloru tła
     dc. w  $8807,$fffe      ; WAIT
     dc. w  $0180,$0333      ; zmiana koloru tła
     dc. w  $8907,$fffe      ; WAIT
     dc. w  $0180,$0111      ; zmiana koloru tła
     dc. w  $8a07,$fffe      ; WAIT
     dc. w  $0180,$0000      ; zmiana koloru tła
     dc. w  $ffff,$fffe      ; koniec CL
```

I to już w dzisiejszym artykule wszystko. Już za miesiąc rozpoczniemy prawdziwe zmagania ...

Marcin „Duddle" Dudar

Własna dyskietka

Ci z Was, którzy posiadali kiedyś C-64 wiedzą jakie kłopoty sprawia początkującym amigowski DOS. W ROM'ie nie ma żadnych rozkazów, ściąganie komend z Workbench'a trwa wieczność. Wszystko jest do tego stopnia skomplikowane, że nawet proste wyświetlenie katalogu dyskietki nastręcza poważny problem. Tych wszystkich użytkowników Amigi, którzy nie rozwiązali do tej pory problemu „własnej dyskietki" zapraszam do przeczytania tego artykułu.

Do pracy potrzebne nam będzie kilka programów. Przede wszystkim konieczny jest dobry kopier, tzw. „file-copier", który potrafi przenosić poszczególne zbiory z dysku do RAM'u i z powrotem na dysk. Do tego celu najlepiej nadaje się Disk-Master 1.3. Jest to program, który oprócz kopiowania ma wiele innych użytecznych funkcji, ale o tym innym razem. Drugim niezbędnym programem jest dowolny edytor tekstu. Zdecydowanie odradzam Workbench'owego Ed'a, gdyż jest on bardzo nieprzyjaźnie nastawiony do użytkownika i najlepszym wyjściem jest zostawienie go w spokoju. Jednym z lepszych edytorów jest „Cygnus Editor"

Bardzo przydatny będzie również programik o nazwie „Set key". Mimo, że wszystkie jego wersje są uszkodzone (mają błędy) i tak jest jedynym programem, który pomoże nam w pracy.

Również przydatny - aczkolwiek niekonieczny - będzie Virus Expert. Dzięki niemu będziemy mogli nie tylko zabijać wirusy (i zarażać nimi dyski...!), ale również zapisywać tzw. Boot-block dysku programem antywirusowym.

Przy kopiowaniu bardzo długich zbiorów, tak długich, że niczym innym nie da się ich skopiować przyda się nam monitor języka maszynowego np. „Timo Rossi Monitor".

Wszystkie inne programy i procedury, które będą nam potrzebne znajdziemy na naszym dysku z Workbench'em.

● **KROK PIERWSZY: formatujemy dyskietkę.**

Tak się dziwnie składa, że producenci dyskietek 3,5" nie wiedzą, że to my - właściciele Amig - je kupujemy. Dlatego też czyste, świeżo kupione dyski są „bezpłciowe" - nie mają żadnego formatu. Pierwszym naszym krokiem będzie więc „powiedzenie" naszej dyskietce, że została wyróżniona z wielu tysięcy jej podobnych i od tej pory będzie pracowała na Amidze. Operacja uświadamiania dyskietki nazywa się FORMATOWANIEM. W czasie jej trwania na dysku zostają zapisane track'i, bloki i wszystkie inne dane (np. nazwa dyskietki), które będą używane przez Amigę. Dyskietkę można sformatować np. z Workbench'a (komenda FORMAT) lub prościej i szybciej z Disk Master'a.

Aby zrobić to pod Workbench'em należy wejść do okna „Shell" i wykonać np:

Format drive df0: name 64+4 noicons

Po chwili komputer poprosi o włożenie dysku przeznaczonego do formatowania i naciśnięcie klawisza „RETURN". W naszym przykładzie „64+4" oznacz nazwę dyskietki i oczywiście można ją dowolnie zmieniać. „Noicons" nie pozwala na zapisanie na dyskietce katalogu Trashcan i jego ikony. Dzięki temu możemy oszczędzić trochę wolnego miejsca.

Tę samą czynność można wykonać również za pomocą Disk Master'a. Z gzymsu (Project) wchodzimy do opcji FORMAT, ustalamy, w której stacji będziemy formatować (klikając myszką na Df1:), wpisujemy nazwę i klikamy na OK! Możemy również włączyć weryfikację (verify) dyskietki - program sprawdzi wtedy poprawność zapisu i jeżeli coś będzie źle - zasygnalizuje to.

W ten sposób nasza dyskietka została uświadomiona co do swojej roli, jaką będzie spełniała w naszych zbiorach.

● **KROK DRUGI : instalowanie dyskietki.**

Jeśli świeżo sformatowaną dyskietkę włożymy do zresetowanego komputera (z „rączką" na ekranie) to nic się nie stanie !

Na dysku nie jest założony tzw. boot. Cóż to jest ? Otóż boot to takie miejsce na nośniku (czyli dyskietce), w którym zapisany jest program uruchamiający się przy starcie komputera (czyli po zresetowaniu). Program taki można zapisać za pomocą komendy INSTALL lub prościej przy pomocy Disk Master'a. W oknie FORMAT zauważyliście na pewno opcję INSTALL. Wystarczy ją włączyć przy formatowaniu, a już dyskietka będzie „zainstalowana" - czyli będzie miała zapisany boot - sector. Teraz już po włożeniu dysku na ekranie pojawi się okienko Amiga - DOS i komputer będzie oczekiwał na wasze rozkazy.

Boot - sector jest jednak miejscem często używanym przez inne - średnio pożądane programy czyli wirusy. W walce z nimi przysłuży się nam Virus Expert. Jego nowszą wersję, poprawioną przez Mac'a / Katharsis mogliście znaleźć na drugim dysku „64 PLUS 4 - Public Domain". Z jego pomocą na dysku można zapisać bardzo dobre boot - bloki, które są programami antywirusowymi. Szczególnie polecam dwa: „Mutant LED" i „KB Checker".

Od tego momentu dyskietka jest już gotowa do pracy i należy wykonać:

● **KROK TRZECI : katalogi.**

```
C       - cd, dir, echo itp.

devs    - clipboards
        - keymaps
```

- printers

fonts - diamond
- helvetica
- diamond.font
- helvetica.font itp.

l - Ram-Handler
- Disk-Validator

libs - diskfont.library
- info.library
- icon.library

s - startup-seguence

To co widzicie powyżej to schemat katalogów przykładowego dysku. Aby ustrzec się bałaganu przyjęto pewien standard zapisu procedur. I tak wszystkie tzw. komendy (Commands) zwyczajowo zapisuje się w katalogu „C". Szuflada „devs" to miejsce na ustawienie klawiszy (mapa klawiatury) - „keymaps", driver'y do drukarek „printers", konfiguracja systemu (system-configuration) itp. W katalogu fonts - jak sama nazwa mówi - umieszcza się kroje czcionek czyli właśnie tzw. fonty. Bardzo przydatne programiki, które powinny być na każdym szanującym się dysku to Ram-Handler i Disk-Validator. Dzięki temu ostatniemu unikniemy błędów przy przepełnieniu dyskietki. Ram-Handler natomiast obsługuje tzw. Ram-disk dzięki któremu będziemy mogli np. kopiować programy.

„Libs" to katalog, w którym umieszczone są biblioteki. Jego zawartość jest uzależniona od zawartości programów na dysku i sposobu ich wywoływania. Jeśli będziemy chcieli korzystać z Workbench'a to należy tam zapisać przede wszystkim biblioteki „icon„" i „info". Jeśli mamy zamiar pracować z różnymi fontami to będzie nam przydatna biblioteka „diskfont".

Ostatnim standardowym katalogiem jest „S". W nim jest umieszczony startup-sequence, ale o tym potem.

- **KROK CZWARTY: system-configuration.**

W katalogu „devs" znajduje się krótki zbiorek o nazwie „system-configuration". W nim zawarte są dane o położeniu ekranu, kolory, wygląd pointer'ka i wiele innych rzeczy. Jak w nie ingerować ? Do tego celu służy program znajdujący się na dysku Workbench-Preferences.

Aby zapisać na dysku swoją konfigurację należy zabootować naszą dyskietkę (patrz słownik slangu), włożyć dysk z Workbench'em i napisać:

Prefs/preferences

Na ekranie zaprezentuje się program, dzięki któremu możemy ustalić swoje preferencje. Teraz wystarczy z powrotem włożyć naszą dyskietkę i wybrać opcję SAVE. System-configuration zostanie zapisany na dysk do katalogu 'devs'.

Zresetujcie teraz komputer i włóżcie naszą dyskietkę do stacji. Komputer samoczynnie przeczyta konfigurację i ustawi ją tak jak to zaplanowaliśmy.Dyskietka wygląda więc coraz bardziej jak nasz własny produkt !

- **KROK PIĄTY: keymaps**

Często się zastanawialiście w jaki sposób rozwiązać ładowanie programów za pomocą naciśnięcia jednego klawisza (np. klawiszy funkcyjnych). Do tego celu służy program SetKey. Z jego pomocą pod każdy z klawiszy waszej Amigi możecie zapisać dowolny tekst. Załóżmy, że na dysku mamy dwa programy, które chcemy uruchomić naciskając F1 i F2. Programy te noszą nazwy np. „DiskMaster" i „Ced".

Wczytujemy SetKey i klikamy na klawisz F1. Na dole ekranu pojawi się ramka z tekstem. Kasujemy to co tam było, piszemy „DiskMaster" i naciskamy klawisz RETURN (pojawi się prostokącik). Teraz klikamy na „Modify keymap". Podobnie postępujemy z klawiszem F2 tyle, że teraz wpisujemy „Ced". Od tej pory klawisze F1 i F2 będą uruchamiały odpowiednio: Disk Master'a i Ced'a.

Zmodyfikowaną „klawiaturę" zapisujemy na dysk (wygodne jest użycie Ram-dysku). Jak teraz uruchomić nasz „keymap" ? Do tego celu służy komenda „SetMap" znajdująca się w „C" na Workbench'u. Wystarczy napisać SetMap gdzie nazwa - to tytuł naszego zbioru zapisanego SetKey'em.

Ciągłe pisanie Setkey... jest jednak męczące. Przydałoby się również wyświetlić jakiś tekst informujący np., że to jest nasz dysk i że są na nim takie a takie programy. Do tego służy:

- **KROK SZÓSTY: startup-sequence**

Co to jest startup-sequence ? Najprościej rzecz biorąc jest to spis programów uruchamiających się po starcie komputera. Do jego tworzenia będziemy używali edytora tekstu.

Jeśli chcemy na przykład ustawić nasz keymap i wyświetlić tekst to zapiszemy:

```
SetMap  nazwa
Type    menu  (Type również znajduje się na WB)
```

Menu to nazwa zbioru (tekstowego), w którym jest zapisany tekst przeznaczony do wyświetlania na ekranie. W ten sposób ożywiliśmy naszą dyskietkę i teraz możemy już wykonać

- **KROK OSTATNI: kopiowanie programów.**

Do tego celu najlepiej nadaje się wspomniany już DiskMaster 1.3. W jednym oknie otwieramy sobie DF0: (source), a w drugim RAM: (destination). Teraz wybieramy myszką odpowiednie programy i naciskamy na COPY. Zaznaczone przez nas zbiory zostaną przeniesione do RAM'u. Teraz wkładamy naszą dyskietkę i klikamy na DF0. Po wyświetleniu katalogu zaznaczamy wszystkie zbiory z RAM'u (ALL) i kopiujemy na nasz dysk (move).

Nasz dysk jest gotowy do pracy !

O kopiowaniu bardzo długich zbiorów przeczytacie za miesiąc.

Sambor Kuźma

LEMMINGS

Wow ! Nowa gra firmy Psygnosis (luty 1991). Trzeba przyznać, że jeszcze nigdy ta firma nas nie zawiodła. Wszystkie gry publikowane przez nią są prawdziwymi Hitami ! Psygnosis kupiła tę grę od mało znanej firmy DMA Design.

Co mamy tym razem ? Otóż jest to historia o przemiłych choć mało inteligentnych stworkach zwanych Lemmingami. Zadaniem gracza jest przeprowadzenie ich przez wiele komnat tak, aby jak najmniejsza ilość utraciła życie.

Jest to rodzaj gry platformowej z tym, że zamiast jednego bohatera mamy ich w porywach aż 100 ! Postacie Lemmingów mimo, że malutkie są świetnie i przezabawnie animowane.

Grafika w tle jest bardzo starannie rysowana. W wielu komnatach nawiązuje do starych gier Psygnosis (Beast, Meneace, Awesome). Podobnie jest z muzyczkami, których jest 21. W komnacie „Revenge of the Beast" komputer gra muzykę z Beast'a, a w „ Meneace " z gry Meneace. Tak częsta zmiana „wystroju" i wiele muzyczek powodują to, że gracz wcale się nie nudzi.

Prowadzenie Lemmingów polega na przydzielaniu im różnych zadań. Robi się to za pomocą myszki. Z normalnego Lemminga (Walker) można uczynić: wspinającego się (Climber), spadochroniarza (Floater), bombowca (Bomber), blokowacza (Blocker), budowniczego (Builder), kopacza tuneli (Basher), górnika (Miner) i kopacza dołów (Digger). Kombinując tymi funkcjami należy przeprowadzić odpowiednią ilość Lemmingów od wejścia do wyjścia. Po drodze czycha wiele pułapek i czasami znalezienie właściwego rozwiązania jest bardzo trudne.

Ciekawym urozmaiceniem jest gra dwóch graczy (potrzebne 2 myszki). Mają oni tę samą grupę Lemmingów, ale każdy ma swoje wyjście. Wygrywa ten, któremu uda się „nakłonić" jak największą ilość stworków do udania się w jego stronę. Szkoda tylko, że nie ma opcji współpracy.

Gra w sumie ma 120 komnat dla jednego gracza podzielonych na 4 poziomy (po 30 w każdym). Pierwszy z nich „FUN" jest tylko wprawką w poznanie zwyczajów Lemmingów. Drugi „TRICKY" jest optymalny. Czasami trzeba sporo pomyśleć jak przejść dany układ. Trzeci poziom „TAXINE" jest już naprawdę trudny (przynajmniej dla mnie). Zupełnie wariacki jest ostatni „MAYHEM". Zauważyli to zresztą autorzy gry, bowiem po przejściu ostatniej komnaty ukazuje się napis: „Niewielu ludzi przeżło poziom MAYHEM. W nagrodę otrzymujesz oklaski od całego zespołu" Po czym pojawia się ekran z całym zespołem DMA Design i słychać owacje na cześć gracza.

W wersji gry dla dwóch graczy do przejścia jest 20 poziomów. W zasadzie cała trudność polega na tym, że co jeden gracz zbuduje to drugi może popsuć. Gdy dosiądzie się dwóch specjalistów - żaden z Lemmingów nie uchodzi żywy...

Całą grę dopełnia świetne intro na początku. Jest to już chyba tradycją, że wszystkie gry Psygnosis mają długie animowane sekwencje przy starcie.

Ocena gry (w skali 0-10)

Grafika	7
Muzyka/FX	6
Animacja	8
Pomysł	10
Przyjemność grania	9
Ocena ogólna	8

Lemmings: DMA Design

Programowanie: Dave Jones

Animacja: Gary Timmons

Grafika: Scott Johnston

Muzyka: Brian Johnston

Testujący:

Hi-Man

Stacje wysokiej gęstości

Kiedy w roku 1935 firma Commodore wypuściła Amigę, jedną z jej rewolucyjnych możliwości była dwustronna stacja 3.5" formatująca dyski na 880kB. W tym czasie wszyscy użytkownicy IBM zmuszeni byli używać dyskietek 5.25" a Apple Mac i Atari ST jako standardowe wyposażenie miały jednostronne stacje 3.5". Nawet kiedy ST i Mac otrzymały stacje dwustronne - mogły one zapisywać tylko 720kB (ST) i 800kB (Mac).

Niestety Amiga długo nie utrzymała się na prowadzeniu. IBM wypuścił nową serię komputerów PS/2 mających stacje 1.44 MB. Podobnie uczyniły Apple i Atari.

Teraz jednak dzięki firmie Applied Engineering Amiga znowu wychodzi na czołówkę! Nowa stacja formatuje dyski na 1.52 Mb bijąc wszystkich pozostałych konkurentów.

Stacja jest produkowana w bardzo ładnie wyglądających obudowach podobnych do zewnętrznych stacji MacIntosh'a. Nie jest to zresztą dziwne: bowiem Applied Engineering robi również stacje dla Mac'a.

Co nowy drive oferuje? Po pierwsze: w środku znajduje się bardzo dobry mechanizm firmy Sony. Może dzięki temu użytkownicy nie będą więcej narzekać na słabe stacje w Amidze.

Ciekawą możliwością jest AUTO EJECT czyli automatyczne „wyskakiwanie" dyskietki ze stacji po zakończeniu pracy. Co prawda system operacyjny Amigi nie ma jeszcze takiej możliwości (dopiero ma ją mieć Workbench 3.0), ale „auto eject" jest i tak przydatny. Na przedniej ściance znajduje się mały przycisk, który służy do wyjmowania dyskietek. Jeśli w czasie pracy stacji (czytanie, zapisywanie) zostanie on wciśnięty drive dyskietkę odda dopiero po zakończeniu operacji z dyskiem! Dzięki temu użytkownicy nie będą narażeni na powstawanie błędów.

Innym ciekawym dodatkiem jest dwukolorowy LED. Świeci on na żółto kiedy drive czyta a na czerwono kiedy zapisuje. W ten sposób użytkownik widzi, kiedy „jakieś paskudztwo" próbuje coś zapisać niepostrzeżenie na dysk.

Kiedy stację podłączymy do komputera zachowuje się ona jak zwykły drive 880kB. Aby móc użyć jej w trybie wysokiej gęstości należy do startup-sequence dodać tylko cztery linie. Specjalny program zmienia Trackdisk-device powodując zgłoszenie się stacji jako dwóch urządzeń:

DF1: jako 880kB i DF5: jako dysk 1.52MB.

Stacja sama sprawdza czy włożona do niej dyskietka jest wysokiej czy normalnej gęstości (sprawdza obecności specjalnego otworu lewej strony dysku HD). Oczywiście można ją oszukać wiercąc dziurę w standardowym dysku. Warstwa magnetyczna na dyskach DD jest jednak o wiele gorszej jakości niż w HD i w ten sposób przerobione dyskietki będą bardzo niepewne.

Stacja pracuje bardzo dobrze ze wszelkiego rodzaju programami kopiującymi. Niestety nie potrafi jeszcze czytać dyskietek 1.44MB w standardzie IBM. Trzeba będzie zaczekać na nadejście odpowiednich programów.

Amiga jest więc na prowadzeniu. Niestety długo się nie utrzyma. IBM zapowiedział już komputer PS/2 Model 90, który będzie posiadał stacje 2.88MB !

Citizen zapowiedział floppy drive 20MB z interface'm SCSI. Tak, tak ! Flappy drive a nie Hard disk ! Wydaje się to być niemożliwe, ale czas pokaże czy Citizen wywiąże się z zobowiązań.

Nowa stacja do Amigi kosztuje obecnie 189$ lecz należy spodziewać się spadku ceny. Kiedy rodzimi handlarze sprowadzą ją na nasze giełdy ? Poczekamy, zobaczymy...

Na podstawie Amiga Computing

HI-Man

Zapraszamy wszystkich do udziału
w stałym konkursie pod hasłem:

Najlepszy program miesiąca

W konkursie udział mogą brać wszyscy, którzy nadeślą własne, nigdzie nie publikowane prace.

Tematyka programów dowolna.

Konkurs rozgrywany jest osobno dla komputerów C-16 i C-64.

Teksty programów należy nadsyłać na adres redakcji na dyskietce lub w postaci czytelnego rękopisu (dyskietki będą przez redakcję zwracane).

Objętość programu wraz z opisem i komentarzem nie powinna przekraczać 4 stron maszynopisu.

Raz w miesiącu Sąd Konkursowy wybierze najlepsze programy przyznając ich autorom dwie główne nagrody po **500.000** zł każda. Decyzje Sądu Konkursowego są nieodwołalne. Oprócz zdobycia głównej nagrody autorzy mają szansę na publikację swych prac na łamach naszego pisma.

Pracę prosimy podpisywać imieniem i nazwiskiem oraz dokładnym adresem autora.

Redakcja

